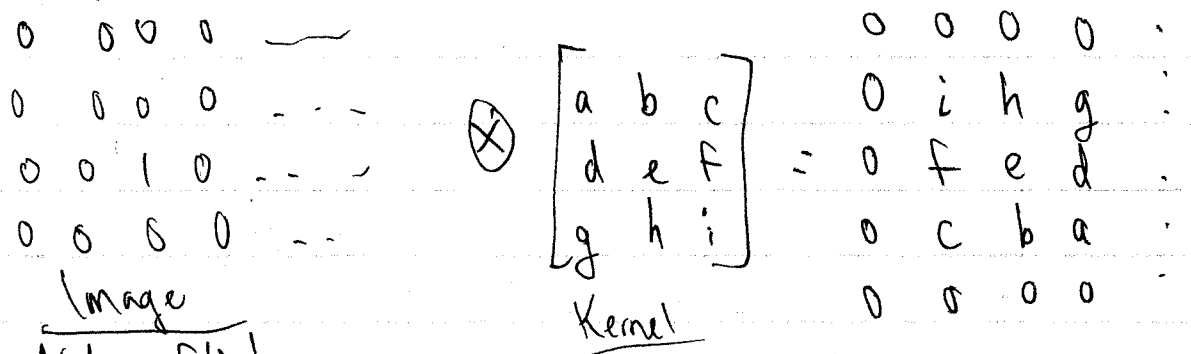
CS6476 (Computer Vision)
Georgia Tech OMSCS

2015-08-24

- Linearity — $H(f_1 + f_2) = H(f_1) + H(f_2)$ - Additivity
  $H(a \cdot f_1) = a \cdot H(f_1)$ - Multiplicative scaling, or homogeneity of degree 1



Cross-correlation

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]\, F[i+u, j+v]$$

$$G = H \otimes F$$

↓ Kernel

Convolution

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]\, F[i-u, j-v]$$

$$G = H * F$$

↑ note ↑ flip

Convolution undoes the flip we'd otherwise see. For symmetric filters: equivalent.

- Convolve filter & impulse → filter
- Convolve image & impulse → image
- Impulse is the identity for convolution!
- Shift invariant: Operator behaves the same everywhere, depends only on immediate neighborhood.
- Convolution = Linear
  - Shift invariant
  - Commutative
  - Associative
  - Identity = impulse
  - Differentiation: $\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g$

    because of linearity & associativity
- "Linearly separable" kernel: You may convolve some column vector & row vector to get it, e.g.

$$\underbrace{\begin{bmatrix}1\\2\\1\end{bmatrix}}_{C} * \underbrace{\begin{bmatrix}1 & 2 & 1\end{bmatrix}}_{r} = \underbrace{\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}}_{H}$$

Now suppose we filter F by H: $H * F = (C * R) * F$

$= C * (R * F)$

But $H * F$ has W^2N^2 multiplications.

$C*(R*F)$ is $WN^2 + WN^2 = 2WN^2$. For large W this can be a huge reduction in complexity!

- Division is linear... (division by a constant that is)
- Boundary issues — Full - start filter far corner at image near corner
  - Same - start filter's middle at corners
  - Valid - filter w/ valid pixels only

2015-08-24b

- near edge — clip filter (black)
  - wrap around (assume periodic image)
  - copy edge/replicate (extend out same value)
  - reflect across edge (reflect image out or 'fold' it)
    - better if statistics on image matter; doesn't introduce an obvious edge
- median filter: edge-preserving, which is useful
  - it is nonlinear however.

2A-L4

- Normalized correlation
  - Scale filter to some property of image (e.g. standard deviation)
- Normalized cross-correlation peaks between two identical signals (normxcorr2)
  - good for template matching (find max)

2A-L5

- Edges - surface normal discontinuity
  - depth discontinuity
  - surface color discontinuity
  - illumination discontinuity
- One way to edge-detect: Find extrema of derivative
- Differential operators
  - Return derivative of some kind
  - Modeled as masks/kernels to compute gradient
  - Threshold this to select edge pixels

_occurs with multivariate functions

Gradient: Vector of partial derivatives

$\nabla f = \left[\partial f/\partial x, \partial f/\partial y\right]$ where f is image

- points in the direction of most rapid increase in intensity

Discrete gradient: Must approximate.

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y) - f(x,y)}{1}$$ "right derivative"

As a correlation filter: $[-1 \quad 1]$, $\begin{bmatrix}1\\-1\end{bmatrix}$ or $\begin{bmatrix}-1\\1\end{bmatrix}$ - depends on origin

$H = \begin{bmatrix}0 & 0\\ -1 & +1\\ 0 & 1\end{bmatrix}$ - This isn't symmetric about image point, and has no 'center' pixel. Instead:

$H = \begin{bmatrix}0 & 0 & 0\\ -1/2 & 0 & +1/2\\ 0 & 0 & 0\end{bmatrix}$ - Average of left & right derivative

- Example: Sobel operator. $S_x = \frac{1}{8}\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$ S_y: and rotate right by 90° for y (if y+ = up)

- Other gradient masks:
  Prewitt, Roberts
- fspecial('sobel') in MATLAB

- Use correlation for gradient: It preserves directionality.
- imfilter by default is correlation!
- imgradient(img, 'sobel')
- In real-world data, noise gives us large derivatives everywhere. So: convolve w/ something like Gaussian first:
  But recall the derivative of convolution:
  $\partial/\partial x\,(h * f) = \left(\frac{\partial}{\partial x} h\right) * f$ - saving us an operation
  (just differentiate the filter kernel!)

2015-08-24c

- But, we find peaks of derivatives with 2nd derivative, and look for zero-crossing.

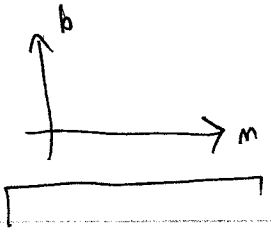
2A-L6

- For derivative-of-gaussian: As with just Gaussian, you vary σ, but here you'll detect finer features with smaller σ, larger with larger σ.
- General gradient → edge steps:
  - smooth derivatives
  - threshold 'significant' gradient
  - "thin" to get localized edge pixels
  - add link/connect edge pixels
- See Canny edge detector for comparison.
- Why you must thin edges: derivatives of slower changes make for larger edges but are just one edge.
- Laplacian: $\nabla^2 h = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$
  - Zero-crossings are the edges!

2015-08-25

2B-L1

- Parametric model: Can represent class of instances, each defined by value of parameters
  - Membership criterion is not local (can't just tell at one point)
  - Computational complexity matters
- We use <u>voting</u> to deal with the complexity of checking for all parameters
  - Each feature, including noise, casts a vote.

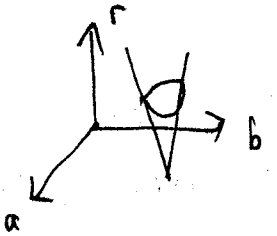


- Line in image space ↔ point in Hough space
- Point in image space ↔ line in Hough space
- Where two lines in Hough space intersect is... a point in Hough space, thus a line from two image points.
  - Then every image point votes on a line...
  - And bin up the points (intersected) to cast votes.
- But really, we use a polar representation ($x \cos\theta + y \sin\theta = d$) because $y = mx + b$ has problems with vertical lines. Line is then (θ, d).
- Hough Accumulator Array - tallies votes
  - Must decide: How big are bins?
- N.B. k^n complexity in space (n dimensions, k bins each)
  Time complexity: Constant with voting elements!
- Their MATLAB example: - Find edges w/ Canny.
- To find line segments is trickier than infinite lines.
- Another approach is to re-apply the Hough transform with finer resolution (smaller bins) in a spot with a lot of peaks
- Using gradient (see slides p30): This is (I think) a way to find lines more quickly and reduce dimensions.
- Extn 2: More votes for stronger edges: This means to find 'stronger' edges by those that are further above threshold.

2015-08-25(b)

$\rightarrow (x-a)^2 + (y-b)^2 = r^2$ (r: known)

- For circle with center (a,b), radius r, a point in image space is a <u>circle</u> in Hough space, of radius r. These will intersect at the center point.
- One approach: Use motion differencing to determine bounding box - and to limit your Hough transform.
- With <u>unknown</u> radius it's a new problem and 3D Hough space. Each image point votes for an entire cone (its surface).



  - But, gradient helps you here.
  - If you know gradient then it votes for just a 3D line, not cone.
- Voting in 3D is a pain. (Exponential explosion)
- Regardless, try to minimize useless votes!
- using gradient helps reduce 'free' parameters
- RANSAC will improve on much of this later

20-L3

- Generalized Hough Transform — Non-analytic models (fixed but arbitrary shape); visual code-word based features - much more recent
  - Visual codeword: uses 'feature patches' that help identify something
- Storing displacements? (not following this really)
- "Hough forests"

$f' = af + b$

$f'(f_L) = \ell = af_L + b$

$f'(f_H) = h = af_H + b$

$\ell - h = a(f_L - f_H)$, $a = \frac{\ell - h}{f_L - f_H} = \frac{h - \ell}{f_H - f_L}$

$\ell = af_L + b$

$b = \ell - af_L$

2C-4

- Basis set, vector space, linear independence, spanning... (not repeating that here), orthogonality
- ~~A basis set needs only to~~
- Right now we treat an image as a point in N x N space, rastered to vector:
  $[x_{0,0}\ x_{1,0}\ x_{2,0} \cdots x_{(n-1),0}\ x_{1,0} \cdots x_{(n-1)(n-1)}]^T$
  With 'normal' basis being vectors:
  $[0\ 0\ 0 \ldots 0\ 1\ 0 \ldots 0\ 0\ 0]^T$
- We can express any image, and the basis is independent, orthogonal, & spans... but isn't useful.
- So: Fourier basis set (defines in terms of faster and slower change)
- $A \sin(wx + \varphi)$ (↑ amplitude, ↑ frequency, ↖ phase)
- Usually for computer vision we don't care about phase (we're not reconstructing the image)
- See p25 for specifics on Fourier transform
- When we do FT, all we're doing is computing a basis set.
- N.B. phase emerges as weighted sum of cos & sin.

- Discrete FT: $F(k) = \frac{1}{N} \sum_{x=0}^{x=N-1} f(x) e^{\frac{-i\,2\pi kx}{N}}$

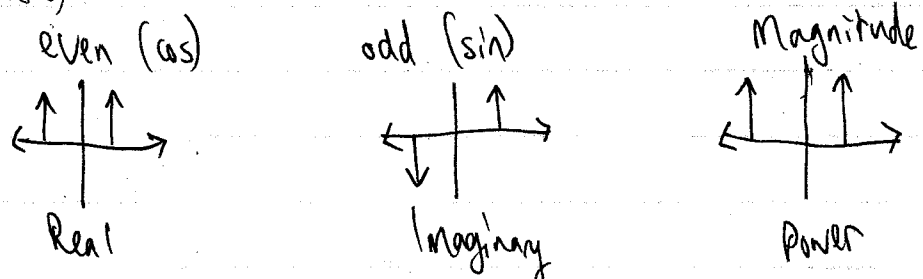
k goes from $-N/2$ to $N/2$ (think of highest freq one can expect)

2015-08-26

$$F(k_x, k_y) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-i \frac{2\pi (k_x x + k_y y)}{N}}$$

↑ 2D discrete Fourier series

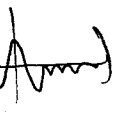
- N.B. this is linear!
- p44: There is no strong horizontal. The vertical center line on the Fourier series is because it 'assumes' that the image is periodic - but it is not (edges do not wrap around - it has a big discontinuity there).



What's this mean? Is this spectra?

2C-L2

- Convolution in spatial domain ↔ Frequency domain multiply
  - See p2,3 on slides
  - Vice versa is also true (spatial multiply ↔ freq. domain convolution)
- Convolution is N^2, while FFT & multiply & IFFT are much faster - which means you can often speed things up thus
- Gaussian kernel, Fourier-transformed, is another Gaussian (but with $1/\sigma^2$ variance, not σ^2)
- If you too abruptly clip higher frequencies, you get ringing (some higher frequencies cancel those out at edges)

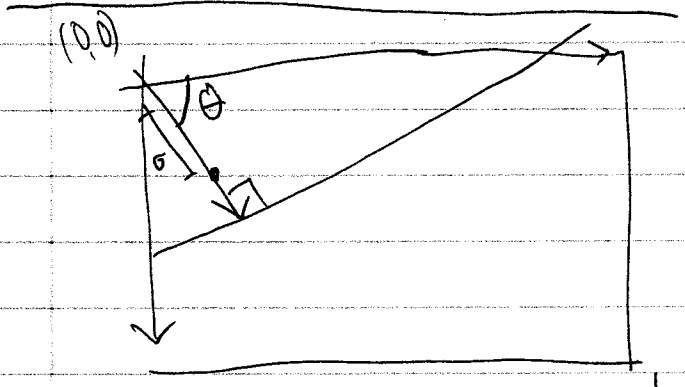
- Note the behavior of a box filter!
- Flip side: Convolve with a sinc function () and get a box filter on frequency side!

- See p10, 11 for some Fourier transform properties
- See Szeliski book too? (p137)

2C-L3

- FT of ~~impulse~~ impulse train → impulse train (but the closer our spatial impulses, the further the frequency impulses).
- also: comb function = impulse train
- $\delta(n)$: 1 if $n=0$, otherwise 0 (Kronecker delta)
- Sampling is just multiplying signal by a comb (thus convolving in frequency)

- Campbell-Robson contrast sensitivity curve



Every line intersects X or Y axis. Some also hit boundaries on other side first.

(x_p, y_p)

Normal form →

$d = x\cos\theta + y\sin\theta$

$x=0$: $y = \frac{d}{\sin\theta}$

$y=0$: $x = \frac{d}{\cos\theta}$

if $x > x_p$:

let $x' = x_p$

$d = x_p\cos\theta + y'\sin\theta$

$y' = \frac{d - x_p\cos\theta}{\sin\theta}$

$y' = y - x_p\frac{\cos\theta}{\sin\theta}$

$y' = y - x_p\cot\theta$

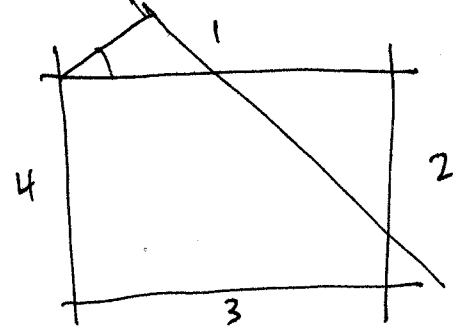
let $y' = y_p$

$d = y_p\sin\theta + x'\cos\theta$

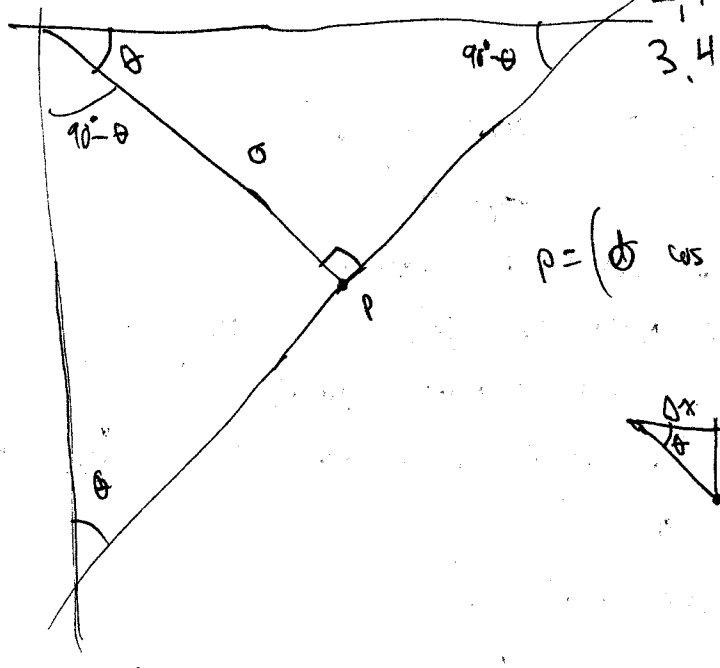
$x' = \frac{d - y_p\sin\theta}{\cos\theta}$

$x' = x - y_p\frac{\sin\theta}{\cos\theta}$

2015-08-31



1, 2
1, 3
1, 4
2, 3
2, 4
3, 4



$P = (\sigma \cos\theta, \sigma \sin\theta)$

$\sin\left(u - \frac{\pi}{2}\right) = -\sin\left(\frac{\pi}{2} - u\right) = -\cos u$

$\cos\left(u - \frac{\pi}{2}\right) = \cos\left(\frac{\pi}{2} - u\right) = \sin u$

$\sin\left(u + \frac{\pi}{2}\right) = -\sin\left(-u - \frac{\pi}{2}\right) = -\sin\left((-u) - \frac{\pi}{2}\right) = \cos(-u) = \cos u$

$\sin\left(\frac{\pi}{2}+u\right) = \sin\left(\frac{\pi}{2}-(-u)\right) = \cos(-u) = \cos u$

$\cos\left(\frac{\pi}{2}+u\right) = \cos\left(\frac{\pi}{2}-(-u)\right) = \sin(-u) = -\sin(u)$

$\sin\left(u+\frac{\pi}{2}\right) = \sin\left(-\left(-u-\frac{\pi}{2}\right)\right) = -\sin\left(-u-\frac{\pi}{2}\right)$

$= -\sin\left(-\frac{\pi}{2}-u\right) = -s$

Notes for future:

- Take better notes in the lecture.
- Use version control!
- Organize into functions from the start.

- How do I detect stronger edges?
  - Larger gradients? (Laplacian?)
- How would I determine length?
  - Seeing how many edge points lie on it?
- How would I find nearby parallel lines?
  - Look for peaks with neighbors with same θ and nearby ρ?
- Creative colorspace conversion?

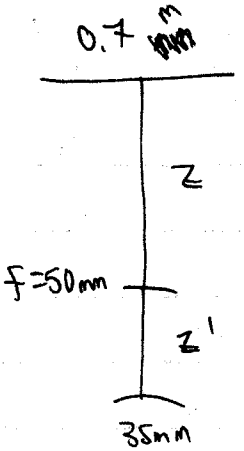
- How can I reduce the time spent keeping LaTeX & Python code in sync?
  (I must keep updating text with parameters I change)

2015-09-04

3A-L1

- Another way to see an image: a 2D projection of 3D points.
- I should take Computational Photography...

C-u
M-q
-refill paragraph
Or:
M-q



$$\frac{1}{z'} + \frac{1}{z} = \frac{1}{f} = \frac{1}{50mm}$$

$$\frac{z}{0.7m} = \frac{z'}{35mm} = \frac{z'}{0.035m}$$

$$z = 20z'$$

$$\frac{1}{z'} + \frac{1}{20z'} = \frac{1}{0.050m}$$

$$\frac{20}{20z'} + \frac{1}{20z'} = \frac{1}{0.050m}$$

$$\frac{21}{20z'} = \frac{1}{0.050m}$$

$$20z' = \frac{21}{0.050m}$$

$$z' = \frac{21}{20}(0.050m) = 52.5mm$$

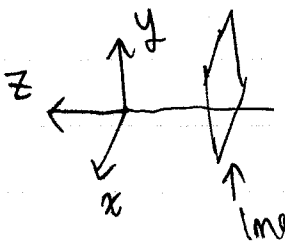
$$z = 20z' = 21(0.050m) = 1.05m$$

- Call it 'perspective effect', not 'perspective distortion'!

 pincushion,  barrel

3A-L2

- We assume a pinhole camera abstraction for most & this
- N.B. 3A-L2 page: 2, 3, 4 (coordinates)
- Center of projection is at <u>origin</u>



Note y is <u>up</u>, not down!
Image plane is in front - image isn't inverted (it's more convenient)

- For this to stay right-handed, Z points in to camera, not out to world. So: Moving 'out' into the world, Z goes more and more negative.
- Homogeneous coordinates...
  - Not linear. Z isn't constant - dividing is non-linear.
  - So: Make it linear by adding a coordinate.

$$(x,y) \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad (x,y,z) \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Don't call the 1 a scale factor! It's not one!

  - You can typically keep homogeneous coordinates around until you actually need an image point
  - Why we love projective geometry: because it's based around linear operations like matrix multiplies.
- Note slide 13: Vanishing point has nothing to do with (x_0, y_0, z_0), the starting point! Only (a,b,c) - the direction - matter. Except if $c \neq 0$.

3B-4

- Stereo is a special case of multiple views
- The question: How do these images relate, and how do they relate to the scene?
- Single views inherently lose structure and depth.
- "Shape from ..." - very popular circa 1980
- A.M. Loh. The recovery of 3D structure using visual texture patterns - find this thesis

2015-09-05

- Even just alternating the two stereo views creates illusion of depth (single eye only, even) (3B-L1 slide 20)
- You can show depth in an auto-stereogram appear by animating it
- Human binocular fusion isn't based on high-level matching, (on large features) but a very low-level process

$$Z = f\frac{B}{x_l - x_r} \qquad \text{Disparity} = x_l - x_r$$

So if disparity = 0, distance is infinite!

3B-L2 → Which textbook chapters correspond to this? (Forsyth & Ponce, ch 7?)

- Epipolar terms:
  - baseline = joins camera centers (centers of projection?)
  - epipolar plane (of a point) - plane with baseline & world point
  - epipolar line - intersection of epipolar plane & image plane (these come in pairs)
  - epipole - point of intersection of baseline & image plane
- All epipolar planes intersect the epipoles
  - → every epipolar line intersects the epipoles. (there really are epipoles, in a pair.)

3B-L3

- Epipolar constraint is a hard constraint, but it is insufficient to give a unique solution.
- Slide 4 - "soft" constraints
- Slide 27 - Reference stereo image & depth
- Slide 30: "left occlusion" = visible to left eye but not to right
  - S_{left} = Left scanline signal, S_{right} = Right

To do:
- Get texts into Calibre
- Copy to e-reader

- Figure out wtf "dynamic programming" is... also, graph cuts

- Diagonal on that graph means the disparity from one pixel to the next stays the same.
- Goal is the least cost path (whatever that means)
  - e.g. Dijkstra's algorithm, dynamic programming
- Slide 35: Smoothness totally ignores image data.
  - ρ = robust mean (?)
- See: Fast Approximate Energy Minimization via Graph Cuts (Y. Boykov, O. Veksler, R. Zabih)
- p38 URL - middlebury.edu site has a lot of 'ground truth' stereo & depth

---

- Assignment 3

$$S_{tx}(r,c) = \sum_{j=0}^{R} \sum_{i=0}^{C} \left[t(i,j) - x\left(r + \overbrace{j}^{i'} - \frac{R}{2}, c + \overbrace{i}^{j'} - \frac{C}{2}\right)\right]^2$$

$$= \sum\sum \left[t(i,j)^2 - 2t(i,j)\, x(i'',j'') + x[i',j']^2 \right]$$

$$= \sum\sum t(i,j)^2 - 2\sum\sum 2t(i,j)\, x(i',j') + \sum\sum x(i',j')^2$$

$$d = \frac{\lambda}{2n \sin\theta} = \frac{500\text{ nm}}{2 \sin\theta} = \frac{500\text{ nm}}{2[\sin 10^{-7}]} = \frac{500 \times 10^{-9}\text{ m}}{2(10^{-7})}$$

$$= 25\text{ m}$$

$t_{text} = 10$ mm

distance = 100 km

$$\theta = \arctan\left(\frac{10\text{ mm}}{100\text{ km}}\right) = \left[5.73 \times 10^{-5}\right]^\circ$$

$$= 10^{-7} \text{ radians}$$

2015-09-09

- How can I meaningfully compare to the ground truth images with unknown units?
  - I must scale my own images to write, also.
  - Is border behavior related to the noise I see at edges?

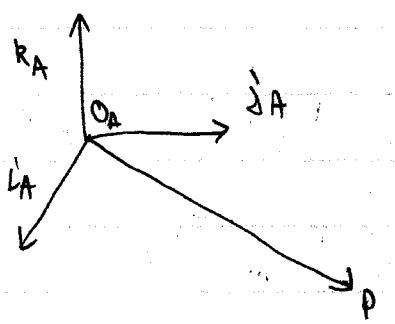
3C-41

2015-09-11

- See Forsythe & Ponce, sections 1.2 & 1.3
- Geometric camera calibration: 1) World coord → camera's 3D coords,
  2) Camera 3D coords → 2D image plane

  1 = extrinsic parameters (camera pose), 2 = intrinsic parameters
- Extrinsic has 6 dimensions (rigid-body)
- F&P notation: ${}^A p$ = coordinates of p in Frame A
  - See slide 13.

$$ {}^A p = \begin{pmatrix} {}^A x \\ {}^A y \\ {}^A z \end{pmatrix} \Leftrightarrow \overrightarrow{OP} = \left({}^A x \cdot \vec{i}_A\right) + \left({}^A y \cdot \vec{j}_A\right) + \left({}^A z \cdot \vec{k}_A\right) $$



Translation:

$$ {}^B p = {}^A p + {}^B(O_A) $$

↑ As origin in B Frame

- If we know P in frame A, and want it in frame B

- Or: $\begin{bmatrix} {}^B p \\ 1 \end{bmatrix} = \begin{bmatrix} I_{3\times3} & {}^B O_A \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} {}^A p \\ 1 \end{bmatrix}$ (Requires homogeneous coords)

- Rotation:

${}^B p = {}^B_A R \, {}^A p$ where ${}^B_A R$ = describing frame A in the coordinate system of frame B

$$= \begin{bmatrix} i_A \cdot i_B & j_A \cdot i_B & k_A \cdot i_B \\ i_A \cdot j_B & j_A \cdot j_B & k_A \cdot j_B \\ i_A \cdot k_B & j_A \cdot k_B & k_A \cdot k_B \end{bmatrix}$$

Note that this is just Frame A's axes, in Frame B.

$$= \begin{bmatrix} {}^B i_A & {}^B j_A & {}^B k_A \end{bmatrix}$$

And vice versa:

$$= \begin{bmatrix} {}^A i_B^T \\ {}^A j_B^T \\ {}^A k_B^T \end{bmatrix}$$

← Both orthogonal matrices!

Inverse = transpose

$$\left({}^B_A R\right)^T = {}^A_B R$$

- N.B. Rotation is not commutative.
- In homogeneous:

$$\begin{bmatrix} {}^B p \\ 1 \end{bmatrix} = \begin{bmatrix} {}^B_A R & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} {}^A p \\ 1 \end{bmatrix}$$

- Total rigid transformation: ${}^B p = {}^B_A R \, {}^A p + {}^B O_A$

Or: $\begin{bmatrix} {}^B p \\ 1 \end{bmatrix} = \begin{bmatrix} {}^B_A R & {}^B O_A \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} {}^A p \\ 1 \end{bmatrix}$

$\left({}^B_A T\right)$ This is invertible!

2015-09-11

$$\vec{{}^{c}p} = {}^{c}_{w}R\ \vec{{}^{w}p} + \vec{{}^{c}_{w}t}$$

- $\vec{{}^{c}p}$ — point now in camera frame
- ${}^{c}_{w}R$ — rotation from world to camera frame
- $\vec{{}^{w}p}$ — point in world frame
- $\vec{{}^{c}_{w}t}$ — translation (world to camera)

$$\begin{pmatrix} {}^{c}p \\ 1 \end{pmatrix} = \begin{pmatrix} {}^{c}_{w}R_{(3\times3)} & \vec{{}^{c}_{w}t} \\ 0\ 0\ 0 & 1 \end{pmatrix} \begin{pmatrix} {}^{w}p \\ 1 \end{pmatrix}$$

with homogeneous c_p, w_p

Extrinsic parameter matrix

3C-L2

- Intrinsic camera parameters:
  - 3D camera coords → 2D image plane (via projection)
  - We did 'ideal' perspective already.
  - Pixels are arbitrary spatial units and we add a scale factor (or scale factors) — $u = \alpha\frac{x}{z}$, $v = \beta\frac{y}{z}$
  - And we don't know origin: $u = \alpha\frac{x}{z} + u_0$, $v = \beta\frac{y}{z} + v_0$ (our optical axis may not be dead-center on our sensor)
  - They might be skew; our horizontal & vertical might be non-perpendicular in actuality

$\vec{p'} \Rightarrow K^c \vec{P}$

homogeneous pixel ← $\vec{p'}$; intrinsic ← K ; Camera 3D ← $\vec{P}$

ugly $K = \begin{pmatrix} \alpha & -\alpha\cot(\theta) & u_0 & 0 \\ 0 & \beta/\sin\theta & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

Simpler $K = \begin{bmatrix} f & s & c_x \\ 0 & \alpha f & c_y \\ 0 & 0 & 1 \end{bmatrix}$ (remove last column)

f = Focal length
s = skew
c_x, c_y = offset

5 DOF

Simplest $K = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- Full extrinsic & intrinsic: $\vec{p'} = \overset{3\times3}{K}\underbrace{\left({}^c_w R \;\; {}^c_w \vec{t}\right)}_{M,\ 3\times4} {}^w\vec{P}$

$$\vec{p'} = M\, {}^w\vec{P}$$

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \simeq \begin{pmatrix} s*u \\ s*v \\ s \end{pmatrix} = \begin{pmatrix} m_1^T \cdots \\ m_2^T \cdots \\ m_3^T \cdots \end{pmatrix} \begin{pmatrix} {}^wP_x \\ {}^wP_y \\ {}^wP_z \\ 1 \end{pmatrix}$$

projectively similar $\longrightarrow$ $u = \dfrac{m_1 \cdot \vec{P}}{m_3 \cdot \vec{P}}$, $v = \dfrac{m_2 \cdot \vec{P}}{m_3 \cdot \vec{P}}$

$$M = \begin{bmatrix} f & s & x_c' \\ 0 & \alpha f & y_c' \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} I_{3\times3} & T_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}$$

intrinsics | projection | rotation | translation

rotation + translation = extrinsics

thus $5+3+3 = 11$ DOFs

2015-09-12

3C-L3

- Calibration
  - Given enough points in world and in image, this can be done directly.
  - This is a homogeneous set of equations but we don't want trivial answer ($m=0$)
  - We want to minimize $||Am||$
    - But m is defined only up to scale...
    - So solve unit vector m^*.

      m^* = eigenvector of A^TA with smallest eigenvalue.

$$A_{2n\times 12} = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ X_n & Y_n & Z_n & 1 & & & & & -u_n & \cdots & & \\ & & & & \cdots & & X_n & \cdots & -v_n & \cdots & & \end{bmatrix}$$

- SVD trick:
  - Find m minimizing $||Am||$ subject to $||m||=1$.

    Let $A = UDV^T$ - that's SVD, D is diagonal, U & V orthogonal. (U & V are norm 1 too)
  - Thus: Minimizing $||UDV^Tm|| = ||DV^Tm||$ and but U is orthogonal & $||U||=1$ so $||m|| = ||V^Tm||$

    and likewise V so
  - So then we're minimizing $||DV^Tm||$ subject to $||V^Tm||=1$
  - Why:
    - Let $y = V^Tm$. Now minimize $||Dy||$ subject to $||y||=1$.
    - But D is diagonal w/ decreasing values.

      So: $||Dy||$ is at minimum for $y = (0,0,0,0,\ldots,0,1)^T$

      (we just want the last column of D)

This is all the direct linear calibration.
Is this DLT as in Zissermann?

- $y = V^T m$ so $m = Vy$ (V is orthogonal)
  - thus: $m = Vy$ is the last column in V.
- & singular values of A are square roots of eigenvalues of A^TA & columns of V are the eigenvectors.

  $A = UDV^T$

  $A^TA = (VD^TU^T)UDV^T$ But U is orthogonal...

  $A^TA = VD^T(U^TU)DV^T = VD^TDV^T$ so $U^T = U^{-1}$ so $U^TU = I$

  $A^TA = V(D^TD)V^T = VD^2V^T$ & D is diagonal so $D^TD = D^2$

  and VD^2V^T is the eigen decomposition where V is the eigenvectors of A^TA. By isolating smallest one from D's decreasing values we're getting smallest eigenvector... thus minimizing $||Am||$, giving us 'm'.
- Another (simpler but worse) approach: (or, inhomogeneous)
  - not gonna copy this. See slides 15 - 16.
- Loosely: This DLT (as all above) doesn't minimize the right error function. Can't impose constraint (e.g. you know focal length very precisely, maybe).
- A better error function is nonlinear & gives distance between 'observed' and 'predicted' image locations
- The "Gold Standard" of calibration is in the book "Multiple View Geometry" (Hartley & Zissermann)
  - Wait a minute, I've done this! Thanks Joe!

2015-09-12b

- M encodes all parameters of camera but we need a way to get them out.
- Let $M = [Q \mid b]$, Q is 3x4, b is column
- Center C is in null-space of projection matrix (proof on slide 26) - so $MC = 0$.
- Thus: $C = \begin{pmatrix} -Q^{-1} b \\ 1 \end{pmatrix}$

2015-09-14

3D-L1

- Translation is special case of projection, preserves lengths/areas, angles, orientation, lines. 2 degrees of freedom.
- Euclidean/Rigid body: doesn't preserve orientation. 3 degrees.
- Similarity transform: doesn't preserve lengths/areas or orientation. 4 degrees
- Affine transform: preserves parallel lines, ratio of areas, lines.. 6 degrees.
- General projective transform: preserves lines, cross ratios (?). 8 degrees of freedom (the homography)
  - 8, not 9, because it's only similarity up to scale.
  - Points that are on a line must stay on a line - or it is degenerate. That is, one can't compute a homography if a line is compressed to a point.

3D-L2

- Point in the image $\Leftrightarrow$ Ray in projective space
- Image reprojection: How to re-project image from same camera center
  - You actually don't need world points for this. It's a sort of 2D image warp.

- Simple use of this: image panoramic / image mosaic
  - Starts from images with same center of projection (but different orientation); find transformation between each image.
  - Images are "reprojected" onto a common to common plane! (this is only good up to 180° FOV).
- Between two orientations is just a homography
- Solving:

$$p' = Hp_1 \qquad \begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Non-homogeneous: Let $i = 1$. Set up $Ah = b$ where $h = [a, b, c, d, e, f, g, h]^T$. So - needs at least 4 points to get 8 equations.
  Solve for h by $\min ||Ah - b||^2$ with least squares and then solve like before...
- Or, use homogeneous and solve with SVD like in the prior lessons.
- Suppose all 3D points sit on a plane...

$$aX + bY + cZ + d = 0$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \simeq \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{11} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \end{bmatrix} \begin{bmatrix} X \\ Y \\ (aX + bY + d)/-c \\ 1 \end{bmatrix}$$

Equivalent to:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \simeq \underbrace{\begin{bmatrix} m'_{00} & m'_{01} & 0 & m'_{03} \\ m'_{10} & m'_{11} & 0 & m'_{13} \\ m'_{20} & m'_{21} & 0 & m'_{23} \end{bmatrix}}_{3\times 3 \text{ homography}} \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

2015-09-14b

- Mapping between planes is a homography... so we can apply this too to 3D points that are on a plane.
- So: You can rectify any rectangular grid you can find in an image.
- Forward warping: The "wrong way" (send every pixel in the unwarped image to a location in the warped image).
- Inverse warping: the "right way" (send every pixel in warped image to a location in the unwarped image - and use interpolation w/ neighbors)

3D-L3

- Projective geometry
  - 2D line: $ax + by + c = 0$ (note scale is irrelevant!)
    $\ell = [a, b, c] \Rightarrow [n_x \; n_y \; -d]$
  - A line is a plane in projective space! Specifically, it's the plane that the normal $\vec{\ell}$ forms (by what it is perpendicular to).
  - Suppose you have 2 points, $\vec{p}_1$ and $\vec{p}_2$, given by their projective points, and you want their line: it's just $\vec{\ell} = \vec{p}_1 \times \vec{p}_2$
  - Or lines $\vec{\ell}_1$ and $\vec{\ell}_2$ and you want their intersection: $\vec{p} = \vec{\ell}_1 \times \vec{\ell}_2$
  - Points & lines are projective <u>duals</u>.
  - Think of points as projective rays to see why cross product gives their line. Think of lines as projective planes to see why cross product gives their intersection point.

- To check if point $\vec{p}$ is on line $\vec{\ell}$: Check $\vec{p} \cdot \vec{\ell} = 0$.
- $p = (x, y, 0)$ is a point at infinity (ray is parallel to image plane)
- If $\ell = (a, b, 0)$ - ~~that is, a that's projective plane is parallel to~~ - line goes through image origin? Normal is parallel to image plane
- In 3D projective geometry, points & planes are duals.
  - Plane is 4-vector

2015-09-17

3D-L4

- To turn the stereo geometry to algebra:
  $X' = RX + T$ (~~X is left camera, X' is right~~)
- Reminder: $\vec{a} \times \vec{b} = \vec{c} \Rightarrow \vec{a} \cdot \vec{c} = 0$ and $\vec{b} \cdot \vec{c} = 0$
- X, X' are projected image points, image points in world coordinates?
- $X' = RX + T$
  $T \times X' = T \times (RX + T)$
  $T \times X' = T \times RX + T \times T$
  $T \times X' = T \times RX$ and dot both sides w/ X':
  $X' \cdot (T \times X') = X' \cdot (T \times RX)$
  $0 = X' \cdot (T \times RX)$
- Another aside:

$$\vec{a} \times \vec{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Let $[a_x] =$ ⌐ (note $[a_x]$ is rank 2)

So $\vec{a} \times \vec{b} = [a_x] b$

2015-09-17 (b)

- Substitute that in: (to $0 = X' \cdot (T \times RX)$)

$X' \cdot ([T_x]RX) = 0$

Let $E = [T_x]R \rightarrow X'^T E X = 0$

E is "essential matrix" and it relates two <u>world points</u>. Since it equals zero, all sorts of multiplications are fine; and so it can apply to other points & image points. (Note that with the help of projective geometry, this can express point & line relationships too).

This can express the epipolar constraint.

? is that right?

2015-09-18

- e.g. Xl where l is a line
- For parallel cameras, $R = I$, $T = [-B, 0, 0]^T$

Then —

$$E = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & B \\ 0 & -B & 0 \end{bmatrix}$$

→ projected points p, p'

~~For cameras~~ $p'^T E p = 0$

$p = \left[\frac{Zx}{f}, \frac{Zy}{f}, Z\right]$.

p' = likewise with x', y'

Divide $p'^T E p$ by Z (valid because it equals 0):

$$[x', y', f] \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & B \\ 0 & -B & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ f \end{bmatrix} = 0$$

$\rightarrow Bfy' = Bfy \Rightarrow y' = y$ (that's our epipolar constraint – horizontal line)

3D-L5

Weak Calibration

- Main idea: Estimate epipolar geometry from redundant set of point correspondences between 2 uncalibrated cameras.

$p^T F p' = 0$

$l = Fp'$ is the epipolar line in the 'p' image associated w/ p'

so: $p^T l = 0 \rightarrow$ p lies on epipolar line

$l' = F^T p$ likewise

- Epipoles are solutions to $Fp' = 0$, $F^T p = 0$
- F is singular 3x3 matrix (it maps between 2D points & 1-D lines)
- F relates pixel coordinates in 2 views, and it is also more general than essential matrix (doesn't need to intrinsic parameters)
- If we estimate, we can reconstruct epipolar geometry.
- Different use: If you have one image & move camera forward, you can compute F for this
- Each point gives one constraint on F $\rightarrow$ but because $p^T F p' = 0$ we can scale F any way we want (so 8 needed)

$$\begin{bmatrix} u' & v' & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = 0$$

Multiply out:

$$\begin{bmatrix} u'u & u'v & u' & v'u & v'v & v' & u & v & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ \vdots \\ f_{33} \end{bmatrix} = 0$$

(but you'll want to stack n of these)

2015-09-18(b)

- Assume we know homography H_{lr} mapping left to right:

  $p' = H_{lr} p$

  Let l' be epipolar line corresponding to p, going through epipole e'.

  $l' = e' \times p' = e' \times H_{lr} p = [e'_x] H_{lr} p$

  But $l' = Fp$ (it's the epipolar line for p)

  So $Fp = [e'_x] H_{lr} p$. But rank of $[e'_x]$ is 2, so rank of $F = 2$.

to solve:

- So for SVD of F, $F = UDV^T$, so last diagonal of D must be 0 to keep rank 2.

$$\hat{F} = U\hat{D}V^T, \quad \text{where } D = \begin{bmatrix} r & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & t \end{bmatrix}, \hat{D} = \begin{bmatrix} r & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- Other neat application: If you shoot stereo images from two unrelated/unaligned viewpoints, you can rectify them with the F matrix to project both views to aligned, parallel stereo views.

- 3D reconstruction also starts here

To-do 2015-09-19:

- Read textbook chapters
- Check over code & submit
- Read next problem set
- Watch lectures!

2015-09-20

4A-L1

- Features
  - A note: Last unit with stereo, you didn't really find features, you just searched along epipolar lines for correspondences.
    To find those epipolar lines you'd need features matched (they were supplied in the assignment).
- Textbook: Forsyth & Ponce, 5.3-5.4; Szeliski, Sec. 4-4.1.1
- Local features: Found in other images,
  found precisely (well-localized),
  found reliably (well-matched)
- Why: Need fundamental matrix to find geometry
  Robotics/vision - detect motion of camera & scene
  Make a panorama!
- Problem #1: Detecting the same point reliably in both images. (Must be repeatable)
- Problem #2: For each point, correctly recognize the corresponding one. (Need reliable & distinctive <u>descriptor</u>)
- We must know where the same feature is in the other images - <u>despite geometric & photometric transformations</u>
- Good features — repeatability/precision
  - saliency/matchability (each has distinctive description)
  - compactness/efficiency (far fewer than pixels)
  - locality (occupies relatively small area around feature, robust to clutter & occlusion).

N.B. This is not the feature vector from Machine Learning

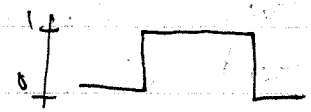
4A-L2

- Corner detection
  - Why: You have gradients in both directions, which restricts your freedom/dimensions
- One of first: Harris corners (or Harris & Stephen corners)

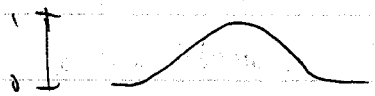
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

w: window function; $I(x+u, y+v)$: shifted intensity; $I(x,y)$: intensity

Window - e.g.



square



Gaussian

u, v should be a small shift; $E \geq 0$ as you move away.
At $(u,v) = (0,0)$ $E = 0$ (no shift)
In a constant image, E is also a constant.
What we want: E should change with a small shift in u, and in v.

- Therefore: 2nd-order Taylor expansion (?)

$$F(\delta x) \approx F(0) + \delta x \frac{d}{dx}F(0) + \frac{1}{2}\delta x^2 \frac{d^2}{dx^2}F(0) \quad \Rightarrow$$

$$E(u,v) \approx E(0,0) + [u,v]\begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2}[u,v]\begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}$$

1st derivs; 2nd derivatives

(for small u & v)

2015-09-206

- For this we need image gradients I_x, I_y, I_{xx},
- See slide 21 (4A-L2)
- ~~Applies for~~ Simplifies to:

$$E(u,v) \approx [u \; v] \, M \begin{bmatrix} u \\ v \end{bmatrix}$$

M is second moment matrix: $M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$

without weight: $\sum \left(\begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \; I_y] \right) = \sum \nabla I \, (\nabla I)^T$

- A 'slice' of $E(u,v)$ has an ellipse and we're looking at a parabola w/ elliptical sections (see slide 28)
- Diagonalization of M: $M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$ where eigenvalues determine ellipse axis lengths
  - Something about M's rank is relevant here...
- Eigenvalues can help classify... (slide 32)
- But... you don't even need to compute eigenvalues.

  Why: $R = \det(M) - \alpha \, \text{trace}(M)^2 = \lambda_1 \lambda_2 - \alpha (\lambda_1 + \lambda_2)^2$

  α is 0.04 to 0.06 (tune it to need)

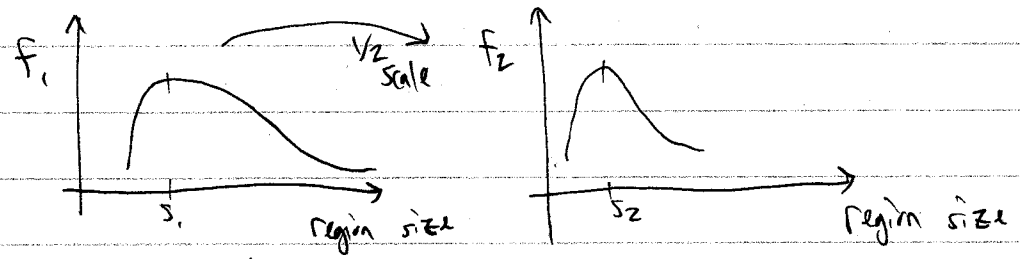
  R <u>depends</u> on eigenvalues but doesn't need to compute them.

  R: large for a 'corner', negative for 'edge', $|R|$ is small for flat function.

4A-L3

Harris

- Why corners: They're rotation-invariant, for one thing.
  Invariant (ish) to intensity - thanks to use of derivative
- But Harris detection is not scale-invariant.
- To get scale invariance:
  - Imagine two circles, of different sizes, on the same (roughly) images but with one scaled. The feature will look the same inside the circles, provided the sizes are chosen right.
  - But how do we choose these circles independently?
  - Consider a simple metric: Average grey area inside circle. (Let f be that.)
  - Image 2 = 1/2 scale of image 1.



  - f_2 is the same as f_1, just scaled.
  - the peaks of these are meaningful for telling us size.
- "good" function gives us one stable, sharp peak.
  ↳ scale detection.
- Scale invariant detection
  - Function is just application of a kernel
  - Laplacian of Gaussian? or Difference of Gaussians
    - Both are invariant to scale & rotation

2015-09-20c

- Key point localization: Find robust extremum both in space & in scale.
- SIFT uses: A pyramid of DoGs to find max values - then eliminate 'edges' and leave corners.
- Process one octave at a time (a doubling in size).
- Look for local extrema across both space and scale
  - Compare each point to 8 neighbors in space, but also 9 neighbors each in the scale above & below
- Next question: How do we match those feature points, now that we've detected them?

4B-L1

2015-09-21

- Descriptors need to be: Almost ~~the~~ the same in both images (invariant)
  - Distinctive
- SIFT (Scale Invariant Feature Detection) (Transform?)
  - Motivation: Harris operator isn't scale-invariant. Correlation (for matching).
  - Goals: Interest operator (detector) that's scale & rotation invariant; A descriptor robust to variations corresponding to typical viewing conditions (This is the most-used part)

- Overall SIFT:

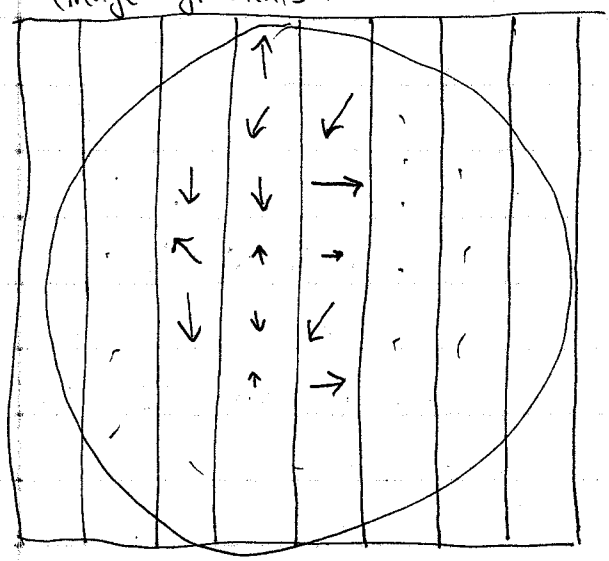
1 - Scale-space extrema detection] Use Laplace-Harris or
2 - Keypoint localization] others
3 - Orientation assignment
4 - Keypoint description

- Orientation assignment: Compute best orientation for each keyplint region.
  - Find "base orientation" - dominant gradient direction
    - Image derivatives then are relative to this.
    - Then, do a histogram of local gradient directions (say, 36 bins) at selected scale (histogram is over each direction)
    - Assign canonical orientation at peak
- Keypoint descriptor
  - Should be:
  - Highly distinctive
  - As invariant as possible to viewpoint & illumination changes

- SIFT vector formation:
  - Feature vector based on histograms of gradients
    - weighted by centered Gaussian...
    - weighted by magnitude of gradient.

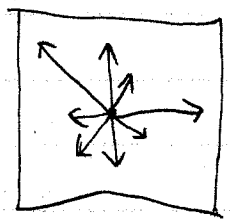
2015-09-21(b)

Image gradients:



↑ Weighted with Gaussian

Now pick a block (say, 4x4) and do a histogram:



8 canonical directions = 8 bins

There are 16 histograms, (in SIFT) so $16 \cdot 8 = 128$ numbers in a feature vector.

It's like a 4x4 grid across this.

- This is sensitive to spatial layout, but not strongly.
- Note that these gradients are already 'normalized' in their orientation.
- After normalization, gradients often need to be clipped to prevent large gradients from dominating
  - And typically one normalizes the histogram so that 'shape' matters, not magnitude.

- A lot of SIFT's features were found empirically (8 canonical directions, 4x4 histograms)
- Also: SIFT is intellectual property of University of British Columbia...
- But, getting SIFT to work is a huge pain.

HB-L2

- Next question: How do we search for a match? (yes, we've still not done this)
  - Nearest-neighbor search is an option, but a slow one.
  - SIFT uses best-bin-first modification to k-d tree algorithm. (slide 6)
  - It's 100-1000 X faster, & works 95% of time as well as nearest-neighbor
- Another way: Wavelet-based hashing (see Szeliski's book)
  - uses Haar wavelets; reduces keypoints by ~ 1000
- Locality-sensitive hashing? (slide 11; Kulis & Grauman)

- 3D object recognition: Training
  - Extract outlines (background subtraction); compute keypoints (interest points & descriptors)
- Testing:
  - Find possible matches.
  - Search for consistent solution (perhaps affine)
- If you find just 3 points, then you can determine affine xform - and for the remaining points you can predict their location.
  - This also means that you don't need all the points, so it's robust with occlusions

2015-09-22

4C-L1

- Robust error functions (for feature-based alignment to find transforms)
- It's a form of "model fitting"
- Loose strategy:
  1. Compute Features, detect, & describe.
  2. Find useful matches (kdtree, best-bin, hashing)
  3. Compute & apply best transformation (affine, transform, homography)

  Or: 3. Loop until happy
  - Hypothesize transformation T from some matches
  - Verify T with other matches

  4. Apply best transform

- We don't really need the "best" match, we just need "good enough" matches.
- Maybe... SSD(patch1, patch2) < threshold?
- Problem: Your "best matches" aren't always right.
- Lowe's method: - SSD of closest match = 1-NN
  - SSD of 2nd closest = 2-NN
  - Now look at how much better 1-NN is than 2-NN.
- If 1-NN and 2-NN are very close then it is probably not a good match.
- You want that match to fall off almost immediately.
- Yes, this will throw out some correct matches.

$||A||^2 = A^T A$?

2015-09-23

- Distinctive vs. invariant competition
  - You are going to get 'outliers' which occur in one and not the other.
- Note that we're "model fitting" (like in Hough transform)
- Typical least squares:
  - Data: $(x_1, y_1) \dots (x_n, y_n)$
  - Line equation $y_i = mx_i + b$
  - Find (m, b) to minimize $E = \sum_{i=1}^{n} (y_i - mx_i - b)^2$
  - This minimizes the vertical distance to the line

$$E = \sum_{\dots} \left(y_i - \begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} \right)^2 = \left\| \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} x_1 & 1 \\ \dots & \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} \right\|^2$$

$$E = \|y - Xh\|^2 \quad \leftarrow \quad y \quad X \quad h$$

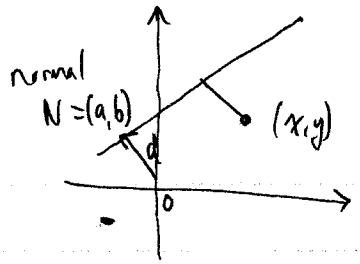
$$= (y - Xh)^T (y - Xh) = y^T y - 2(Xh)^T y + (Xh)^T (Xh)$$

- derivative w.r.t. vector

$$\frac{dE}{dh} = 2X^T Xh - 2X^T y = 0$$

$$X^T Xh = X^T y \Rightarrow h = \underbrace{(X^T X)^{-1} X^T}_{\text{pseudoinverse}} y$$

- But this is problematic since it handles only vertical error. We want 'total least squares'.



2015-09-23(b)

- Total least squares
  - Distance between point (x_i, y_i) and line $ax+by=d$.
  - Find (a, b, d) to minimize sum of squared perpendicular distances. - $E = \sum (ax_i + by_i - d)^2$

$\partial E / \partial d = \sum -2(ax_i + by_i - d) = 0$, $d = \frac{a}{n}\sum x_i + \frac{b}{n}\sum y_i$

$d = a\bar{x} + b\bar{y}$ where $\bar{x}$ is mean of x

~~- Get this later from slide.. (08 video)~~

- Then $E = \sum (a(x_i - \bar{x}) + b(y_i - \bar{y}))^2 = \left\| \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \right\|^2$

(U = the matrix, h = the vector)

$E = (Uh)^T(Uh)$ & solve $dE/dh = 2(U^TU)h = 0$ for non-homogeneous; again SVD is least-squares solution

Solve $(U^TU)h = 0$ under $\|h\|^2 = 1$

- Why perpendicular distance?
  - Our model is of a line where points are corrupted by Gaussian noise perpendicular to line
- Doesn't work so great with non-Gaussian noise.
- Problem: Squared distances heavily penalize outliers.
- "Robust estimators"
  - Minimize $\sum \rho(r_i(x_i, \theta); \sigma)$

    $r_i(x_i, \theta)$ = residue of ith point w.r.t. model params θ

    ρ = robust function with scale parameter σ
  - One example: $\rho(u; \sigma) = \frac{u^2}{\sigma^2 + u^2}$ - acts like squared distance in small range but saturates w/ larger u (σ determines how quickly).
    - This is just one robust estimator of many.

4C-L2

- 'Finding consistent matches'
  - Some best matches are correct! Some... aren't, and they aren't part of any consistent model we're fitting.
  - So: RANSAC (Random Sample Consensus)
    - Fitting a model is easy if we know which points belong & which don't.
    - If we proposed a model, we can easily guess which points are 'inliers' which probably belong.
- RANSAC: Randomly pick points to define model. Repeat until you find a good model (one with many inliers).
  - this copes with a large proportion of outliers.
- Algorithm: - Sample randomly the # of points needed to fit model.
  - Solve for model parameters with these points
  - Score by the fraction of inliers within some threshold. (Get a consensus set C_i)
  - Repeat until best model is found w/ high confidence.
- A model type has a "minimal set" - number of points needed to describe. For a line, it's 2.
  For translation, it's just a pair of matched points.
  Homography (for plane) - 4 point pairs.
  Fundamental matrix - 8 point pairs. (Actually, 7, but let's ignore that for now, because F is rank 2, not 3...)

2015-09-23c

- Choosing parameters:
  - Initial # of points for minimal set s (typically minimum needed for model fitting)
  - Distance threshold t
- Distance threshold:
  - Assume location noise is Gaussian w/ σ^2
  - Distance d has Chi distribution with k DOF where k is dimension of Gaussian. For distance off a line, k = 1.

$$f(d) = \frac{\sqrt{2}\, e^{-d^2/(2\sigma^2)}}{\sqrt{\pi}\,\sigma}, \quad d \geq 0$$

From ↓
- Your noise model, you can make a distance model. From this you can pick a threshold.
- e.g. For 95% cumulative threshold t: $t^2 = 3.84\sigma^2$. That is: $t^2 = 3.84\sigma^2 \Rightarrow$ 95% probability that $d < t$ when point is inlier.

- Next parameter: Number of samples N. (How many models do we solve for?)
  - Choose so that, w/ probability p, at least one random sample set is free from outliers
  - Set N based on outlier ratio e.
    - s = # of points to compute, p = prob. of success, e = proportion of outliers, so % inliers = (1 − e)
    - P(sample set w/ all inliers) = $(1-e)^s$ by combinatorics (assuming # of points is very large).
    - P(Sample set has ≥ 1 outlier) = $1 - (1-e)^s$
    - P(all N have outlier) = $(1-(1-e)^s)^N$. We want this < 1 − p

? This is the proportion throughout all of the points.

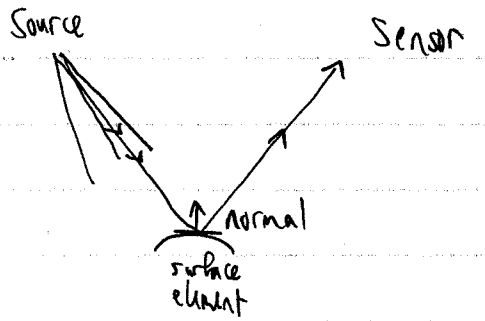
So: $(1-(1-e)^s)^N < (1-p) \Rightarrow N > \frac{\log(1-p)}{\log(1-(1-e)^s)}$

- This increases steeply with s, but less so with e.
- This is a hallmark of RANSAC: It copes well with a fairly high number of outliers.
- Another cool feature: that formula never involves how many points ("putative matches") you have – only how many are in your minimal set, and what proportion of outliers
- Once you have inliers via RANSAC, you may compute a better model by using all the inliers (and averaging or something to find an optimal model).
- See slide for "adaptive procedure" to guess e value and thus choose N.
- Why RANSAC is good:
  - Simple & general. Works well in practice and on many different problems.
  - Copes well to many outliers.
  - Much better than Hough transform when you have many parameters, and easier to tune than Hough
- Why it's bad:
  - Computational time grows quickly with # of parameters.
  - Not good w/ multiple fits, or w/ approximate models.

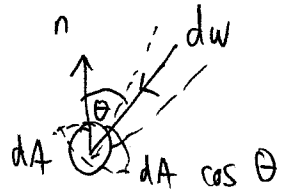
2015-10-02

SA-L1

- Photometry (measuring light)
  - A lot of this is already well-understood from computer graphics.
- 20-40 years ago, computer vision was just seen as the reverse problem of computer graphics (if you have this image with this shading, work backwards & find object/object properties/lighting
- In the slide 3 image, we see a road texture, plus a shadow (we don't see it as part of the texture), plus a measuring tape (part of neither)
- Likewise we perceive slide 4 images as having reflections - not real items behind glass
- Nowadays, computer vision isn't seen so much as a sister field of computer graphics - but as also having strong ties to machine learning as a data-driven inference problem.
  However, an understanding of the physics of light still is critical.
- Surface appearance

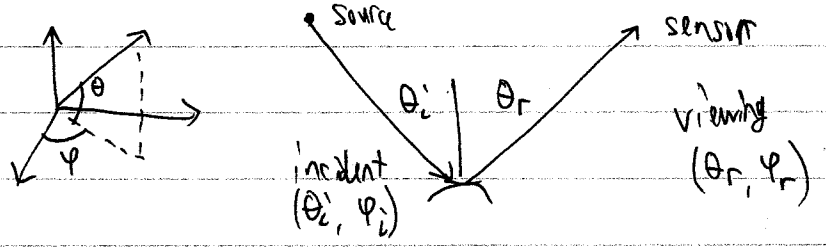


  - Image intensity = f(normal, surface reflectance, illumination)
  - Surface reflectance depends on both viewing & illumination directions



Radiometry

- Radiance (L) - Energy that a ray carries
- Power per unit area <u>perpendicular</u> to direction of travel, per unit solid angle
- Units: Watts per square meter per steradian ($W m^{-2} sr^{-1}$)
- Irradiance (E) - Energy arriving at surface
  - Incident power in a direction per unit area
  - Units: $W m^{-2}$
- Think of a lamp shining on a circle on a card. Tilt the card somewhat, and because of the foreshortening, less light reaches that same-sized circle.
- $E(\theta, \varphi) = L(\theta, \varphi) \cos\theta \, dw$ - surface receiving radiance L, coming in from dw, 'sees' irradiance E
- BRDF: Bidirectional Reflectance Distribution Function



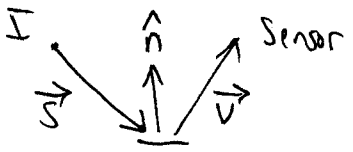
$$\text{BRDF: } f(\theta_i, \varphi_i, \theta_r, \varphi_r) = \frac{L^{\text{surface}}(\theta_r, \varphi_r)}{E^{\text{surface}}(\theta_i, \varphi_i)}$$

- That is: Ratio of incident radiance to reflected irradiance.
- $f \leq 1$ (it can't produce more light)
- Helmholtz reciprocity: $f(\theta_i, \varphi_i, \theta_r, \varphi_r) = f(\theta_r, \varphi_r, \theta_i, \varphi_i)$
  - Or: Light doesn't know which way it's going (to or from the source or the sensor)

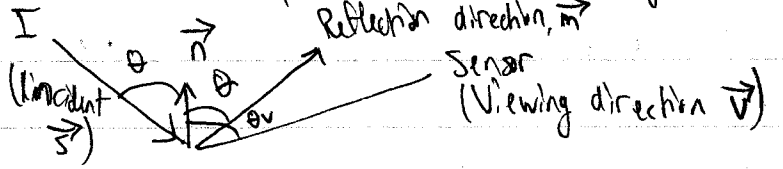
2015-10-02b

- Also, isotropy/rotational symmetry:
  $f(\theta_i, \varphi_i, \theta_r, \varphi_r) = f(\theta_i, \theta_r, \varphi_i - \varphi_r)$
  - That is, if we rotate the surface around its normal, the BRDF doesn't change (at least, in our case, we assume this property)
- Reflection models
  - "body reflection" - light scatters around top layer of surface and is reflected in all directions
  - Diffuse reflection is this; it has a matte appearance.
  - It looks the same from every direction
  - Specular reflection is "surface reflection" - light reflects directly from surface and gives a glossy reflection with highlights
  - Many metals behave like this (it's due to electron interaction)
- Image intensity = body reflection + surface reflection
- Lambertian model: Only body reflection
  - ~~Only the incident light matters - not the direction of the reflected light.~~
  - N.B. In Lambertian, the amount of light reflected is <u>not</u> the same from every angle! It falls off with the cosine of the angle from the normal. But, when you view at that angle, you take in a larger <u>area</u> as that angle grows, and it cancels out.

See slide 26



- So: Lambertian BRDF doesn't reflect the same amount of light from every angle. It <u>looks</u> the same at every angle because the steeper angles cover a larger area of the surface.
- So, BRDF is a constant — the <u>albedo</u>, ρ_d.
  Surface radiance $L = \rho_d I \cos\theta_i = \rho_d I (\vec{n} \cdot \vec{s})$
  ↑ source intensity
- Back to specular reflection
  - In a mirror, reflection occurs <u>only</u> at mirror angle.



So $L = I \rho_s \delta(\theta_i - \theta_v)\delta(\varphi_i + \pi - \varphi_v)$

Recall that $\delta(x) = 0$ except if $x = 0$, where $\delta = 1$

That is, you must be <u>viewing</u> (θ_v, φ_v) at the incident angle θ, and <u>across</u> φ.

To abuse notation a little:

$$L = I \rho_s \delta(\vec{m} - \vec{v}) \text{ or } I \rho_s \delta(\vec{n} - \vec{h})$$

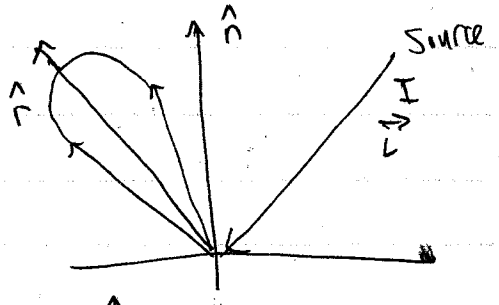
If $\vec{m}$ = reflection direction, $\vec{v}$ is viewing direction, this says that $L = 0$ except where $\vec{m} = \vec{v}$.

$\vec{h}$ = half angle = angle halfway between $\vec{s}$ (the incident angle) and $\vec{v}$ (the viewing angle) — this must exactly be the normal for the delta function to be 1.

- But... most things aren't perfect mirrors. It spreads out past the mirror point.

2015-10-02c

- So: $L = I \rho_s (\vec{m} \cdot \vec{v})^k$



↑ the value of k controls that lobe's size

- Phong reflection: BRDF of Lambertian + Specular

5B-L1

- Lightness
  - f(normal, surface reflectance, illumination) = image intensity. We can't really solve for one parameter without making assumptions about others. It's an ill-posed problem. → lightness that camera sees
- Planar Lambertian material: $L = I \times \rho \times \cos(\theta)$
  I = illuminance, ρ = reflectance (albedo), θ = angle between light and $\vec{n}$
- If we combine θ & I at a point into $E(x,y)$ and have reflectance function $R(x,y)$...
  $L(x,y) = R(x,y)\,E(x,y)$
  We can see $E(x,y)$ as how a white piece of paper would look under this illumination.
  $R(x,y)$ can be seen as a constant planar patch of light (???) —

- But if we measure L, and we want R without knowing E - how?

"the Mondrian world"

- Assumptions (computationally):
  1. Light varies slowly.
  2. Within an object or patch, reflectance is constant.
  3. Between objects, reflectance varies suddenly.
- Formally: Assume illuminance $E(x,y)$ is low-frequency. R is constant over patches, separated by edges.

- Edwin Land (1909-1991) - invented Polaroid Land camera
  - Made Land's Retinex theory

See Forsyth book.

  - Goal: Remove slow variations from image
  - One approach: log of both sides. ($\log L = \log E + \log R$) ← note + not *
    - Hi-pass filter (use derivative?)
    - Threshold to cut small low frequencies
    - Invert process; take integral, exponentiate
  - See slide 25 example
  - It's a little trickier to do in 2D. (Does GIMP do this?)
- These approaches fail anyplace your shape varies suddenly
- Human color & lightness constancy
  - Color constancy: determine hue & saturation under different lighting colors
  - Lightness constancy: gray-level reflectance under differing intensity of lighting.
- Humans are good at the above. We are not good at determining colors & levels of light.

2015-10-02d

- The point of a white balance card: To tell a camera's algorithm that any color seen on the card is due to the <u>illuminance</u> - not the reflectance.

2015-10-03

5C-L1

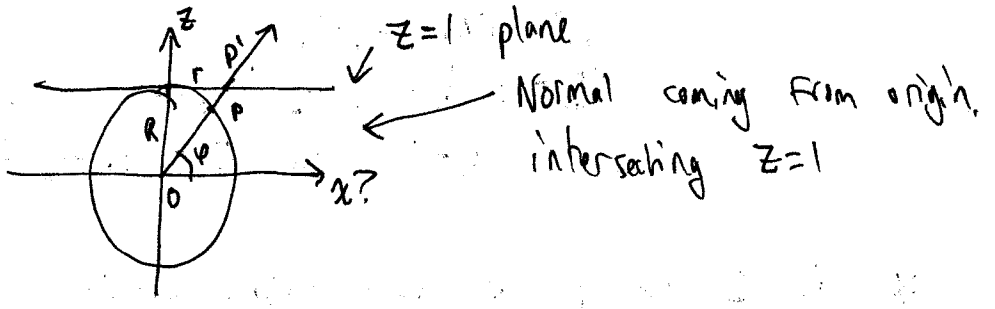
- Shape from shading
  - How do intensity & shape relate?
- Surface normal
  - Assume surface $z(x,y)$. Let $p = -\partial z/\partial x$, $q = -\partial z/\partial y$. (Some books use positive rather than negative).
  - For a surface point, define 2 tangents.

$$t_x = (1, 0, -p)^T, \quad t_y = (0, 1, -q)^T \qquad \text{(not unit vectors)}$$

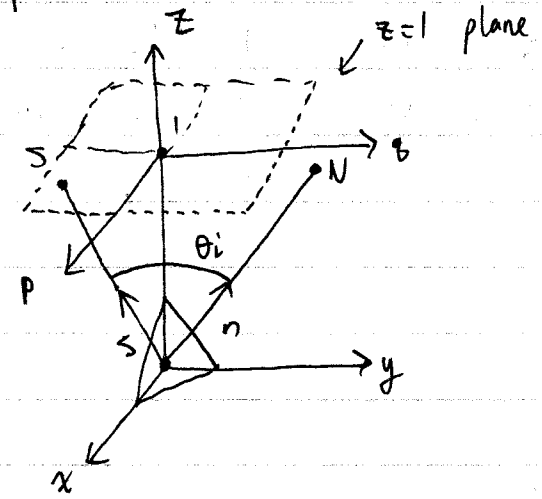
$$\hat{n} = \frac{N}{\|N\|} = \frac{t_x \times t_y}{\|t_x \times t_y\|} = \frac{1}{\sqrt{p^2+q^2+1}}(p, q, 1) \qquad \text{(our normal)}$$

- The <u>Gaussian sphere</u> represents all possible surface normals. That is, every normal has a point on it.
  - From (p,q) we have a projective mapping.

In 2D: (side view of Gaussian sphere)



- $z=1$ plane = <u>gradient space</u> and every normal has a point here



s = Source vector
n = Normal
θ_i between
S = intersection of s w/ $z=1$
N = likewise n

$$n = N/\|N\| = \frac{(p,q,1)}{\sqrt{p^2+q^2+1}}$$

$$s = S/\|S\| = (p_s, q_s, 1)/\sqrt{p_s^2+q_s^2+1}$$

- But in a Lambertian surface, only θ_i matters, and we need its cosine.

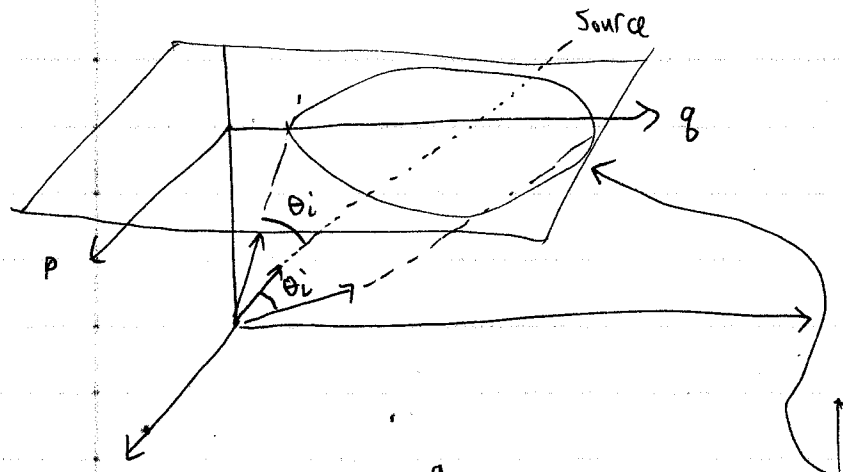
$$\cos\theta_i = n \cdot s = \frac{pp_s + qq_s + 1}{\sqrt{p^2+q^2+1}\sqrt{p_s^2+q_s^2+1}}$$

- Note that this (particularly z as a function of (x,y)) is a very viewpoint-specific view of the scene.
- Reflectance map: Relates image brightness $I(x,y)$ to surface orientation (p,q) for given source direction & surface reflectance
  - Lambertian: k = source brightness
    ρ = surface albedo
    $I = \rho \cdot k \cdot \cos\theta_i = \rho \cdot k \cdot (n \cdot s)$
    Let $\rho \cdot k = 1$, then $I = \cos\theta_i = n \cdot s$

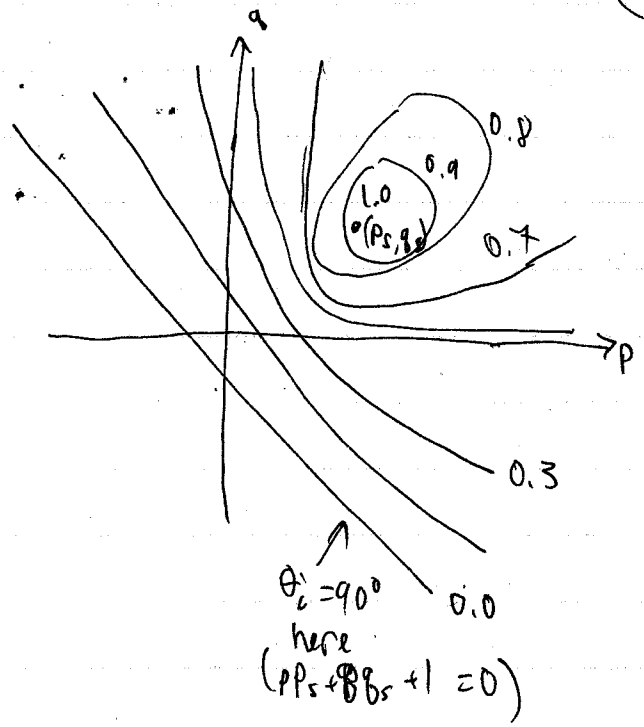
* N.B. I use p for both surface albedo, and (p,q). Sorry.

2015-10-03(b)

So... $I = \cos\theta_i = n \cdot s = \dfrac{pp_s + qq_s + 1}{\sqrt{p^2+q^2+1}\sqrt{p_s^2+q_s^2+1}} = R(p,q)$



The side lines are the cone of constant θ_i, and they form an "iso-brightness" contour in pq-space



R(p,q) is at max. where $(p,q) = (p_s, q_s)$

- But given R(p,q) (that is, (p_s, q_s) and surface reflectance) we can't determine unique (p,q) - we can only determine a location down to a specific iso-brightness contour.
- So, we need additional constraints.

→ "It works poorly in the lab, and not as good elsewhere."

- More constraints (but doesn't really work): Shape from shading. The assumptions made are quite restrictive.
- More images (photometric stereo)
  - Same object, same pose, different lighting
    → $I = \rho \cdot k \cdot \cos\theta_i = \rho n \cdot s \quad (k=1)$
    - Say we have 3 light sources. Assume Lambertian

$$I_1 = \rho n \cdot s_1$$
$$I_2 = \rho n \cdot s_2$$
$$I_3 = \rho n \cdot s_3$$

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \rho \begin{bmatrix} s_1^T \\ s_2^T \\ s_3^T \end{bmatrix} n$$

    - We can solve this easily for $\rho\hat{n}$, and it readily works with more lights.
- Geometrically: We're moving (p_s, q_s) around in (p,q) space - and intersecting iso-brightness contours.
- This actually works somewhat in practice!
  - See www.trimensional.com

6A-L1

- Introduction to motion
  - Video = sequence of frames captured over time - usually quickly.
  - Then, $I(x,y,t)$ - also a function of time.
  - But, t isn't the same as x & y despite being just another dimension.
- Motion & perceptual organization
  - Started w/ gestalt psychology (Max Wertheimer)
    - What he saw was the way that a bunch of spaced lights convey something like 'a moving train' to us, despite just being independent lights

2015-10-03c

- Motion analysis applications: - Segment object in space & time
  - Estimate 3D structure
  - Learn dynamical models (how things move)
  - Recognize events & activities
  - Motion stabilization

- Motion estimation techniques:
  - Feature-based methods (we're already familiar)
    - Extract visual features (corners, textures) and track across frames.
    - This is sparse (it only gets motion at specific points visible across frames) but more robust.
    - Suitable for large image motion - 10s of pixels
  - Dense, direct methods
    - Directly recover motion of each pixel by "spatio-temporal image brightness variations"
    - Dense motion field - sensitive to appearance variation though.
    - Suitable for video (particularly w/ higher framerates) and when motion is small

We're going to leave this, as we've covered it somewhat, and not discuss it for motion.

6B-L1

- Dense flow: Brightness constraint
  - "Optic flow" - The <u>apparent</u> motion of objects or surfaces.
    (<u>Actual</u> motion is the "motion field" but we don't handle that here.)

- We want: pixel motion from $I(x,y,t)$ to $I(x,y,t+1)$
  - Need to solve pixel correspondence problem — need to look for nearby pixels of same color between t & $t+1$.
  - We assume: - color constancy (or brightness constancy for greyscale) - a point in $I(x,y,t)$ looks the same in $I(x',y',t+1)$
    - small motion (points don't move far)
- Say that point moves by (u,v).

Then: $I(x,y,t) = I(x+u, y+v, t+1)$
$\Rightarrow \quad 0 = I(x+u, y+v, t+1) - I(x,y,t)$] brightness constancy

If we assume small motion (say, $u,v < 1$ pixel, or smooth):

- Taylor series expansion:

$$I(x+u, y+v) \approx I(x,y) + \partial I/\partial x\, u + \partial I/\partial y\, v$$

Combining: $0 \approx I(x,y,t+1) + I_x u + I_y v - I(x,y,t)$

where $I_x = \partial I/\partial x$ at t or $t+1$ - doesn't matter, we consider motion too insignificant

So: $0 \approx [I(x,y,t+1) - I(x,y,t)] + I_x u + I_y v$

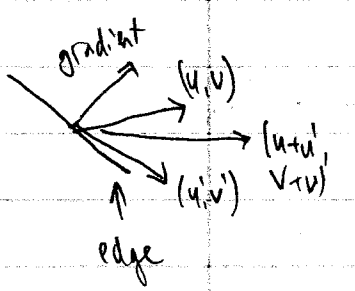
$0 \approx I_t + I_x u + I_y v$

I_t is the temporal derivative, and also:

$0 \approx I_t + \nabla I \cdot \langle u, v \rangle$

As u & v approach zero this is exact:

$0 = I_t + \nabla I \cdot \langle u, v \rangle$



This is two unknowns but only one equation.

Why: The component of flow in the gradient direction is determined. The component parallel to an edge is unknown.

2015-10-03d

- This is called the <u>aperture problem</u>:
  - When seeing motion of something through an aperture (that only lets you see part of an edge), you only see motion along its gradient. If motion is along the edge itself, you see no motion through the aperture
  - N.B: The barber-pole illusion has to do with the aperture problem - but it doesn't explain it!
- Why this matters here: Brightness constancy formula deals only with <u>one pixel</u>.
- Additional flow constraints
  - Local: Nearby pixels flow together
  - Global: Motion must be consistent over image (as a soft constraint)
- We still need a way to solve for both u & v.
- One way: (to add a constraint)

$$e_c = \iint_{image} (I_x u + I_y v + I_t)^2 \, dx \, dy$$

(error in optical flow constraint)

- Smoothness constraint: $e_s = \iint_{image} (u_x^2 + u_y^2) + (v_x^2 + v_y^2) \, dx \, dy$
  - Penalizes <u>changes</u> in u & v; they shouldn't change greatly.
- Then <u>solve</u>, minimizing $e = e_s + \lambda e_c$ → weighting factor

  <u>solve</u> for (u,v) at each image point

  This ends up being difficult, but not impossible, to solve.

Dense Flow: Lucas and Kanade

6B-L2

- The idea: Impose <u>local</u> constraints to get more equations per pixel.
- One way: Assume that all (u,v) values are the same over a window

$$0 = I_t(p_i) + \nabla I(p_i) \cdot [u \; v]$$

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}}_{A_{25\times 2}} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{d_{2\times 1}} = \begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

- Then use normal least-squares, minimize $\|Ad - b\|^2$

$$(A^TA)\,d = A^Tb$$

$2\times 2 \quad 2\times 1 \quad 2\times 1$

(Σ is over your window)

$$\begin{bmatrix} \Sigma I_x I_x & \Sigma I_x I_y \\ \Sigma I_x I_y & \Sigma I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \Sigma I_x I_t \\ \Sigma I_y I_t \end{bmatrix}$$

- This is solvable if A^TA is invertible.
  - Then they must be well-conditioned
    - λ_1/λ_2 must not be too large (eigenvalues)
- This is actually identical to in the Harris corner detector (slide 14)
- But we're working with RGB, so 3 times as many equations? Not exactly, since they're very correlated.
- If motion is large then Taylor expansion doesn't hold. We can use iterative refinement in this case.

2015-10-04

LK

- Iterative refinement: (Lukas-Kanade Algorithm)
  - Estimate velocity at each pixel by solving Lucas-Kanade equations
  - Warp $I_t \rightarrow I_{t+1}$ with estimated flow field
  - Repeat until convergence!
- The only real addition here is the warping step.
- Low-pass filters also help here

6E-L3

- Hierarchical LK
- Errors in LK: - Brightness constancy doesn't hold
  - Point doesn't move like neighbors (motion segmentation?)
  - Motion is too large — Iterative refinement
    - Local minima: Coarse-to-fine estimation
- Aliasing can occur in optical flow too. Larger motion can masquerade as smaller motion (see slide 4)
- Solution: Reduce the resolution. (slide 5)
- Gaussian pyramid (slide 6) (and 15...)
- This eventually brings the motion to be sub-pixel.
- This is a <u>multi-scale</u> approach.
- At highest level: Run iterative L-K, warp & upsample, then go to next (finer) level.

- Gaussian pyramids
  - Let you do both low-pass & band-pass filtering
  - Subtracting adjacent images gives you... the Laplacian (see slide 16) - "subband" images.
    They're approximately bandpass images.
- Interestingly: If you have smallest Gaussian image (coarsest) and the whole Laplacian pyramid, you can construct the original image.
- Slides 21, 22 for the filters needed here
  - Note that these are separable filters!
- I need to look over all this again... (Are they in textbook?)
- Slide 23 does some weird seamless transition between apple & orange, by blending just the coarser pyramids

- Sparse LK
  - Hierarchical LK but applied only to 'good' feature locations

2015-10-07

§8-L4

- Motion models
- Slide 9: Z is only at the translation term → motion parallax
  because depth has no impact on the image plane rotating
  - Panoramic stitching uses this principle

2015-10-07 (b)

- Layered motion (Layered Representation for Motion Analysis, Wang & Adelson)
  - Break images up into 'layers' with coherent motion

2015-10-18

7A-L1

- Introduction to tracking
  - Objects are moving over time and we want to keep track of them
  - e.g. tracking a person moving around; tracking a leaf on a tree as wind blows & camera moves, pans, moves
  - tracking multiple people in the street walking around
  - The usual face tracking
- So far we've only tracked (ish... with optical flow/RANSAC) between pairs of images.
  With more images, or larger displacements, maybe we need to handle dynamics. Occlusion can also be an issue, and drift (e.g. we gradually start tracking something else).

- Shi-Tomasi feature tracker - only compute motion where you should
  - Frame-to-frame, track w/ LK and pure translation
  - Check consistency of tracks by affine registration to the first (or earlier) observed feature instance
    - Affine = better for large displacements; checking w/ first lowers drift

J. Shi and C. Tomasi, "Good Features to Track". CVPR 1994.

- Tracking with dynamics:
  - Given model of expected motion (e.g. constant velocity), predict object location & restrict search based on that.
  - Prediction is difference between tracking, and just detection.
  - Detection is independent for each frame
  - Tracking: We predict new location in next frame with estimated dynamics - then we update with new measurements.
- Goals: do less work looking for object - restrict search
  - Improved estimates since measurement noise is tempered w/ smoothness & prior dynamics.
- Assumptions: Continuous motion, objects don't disappear and reappear, camera doesn't move instantly, camera pose changes gradually.

7B-L1

- Tracking as inference
  - X = hidden state (true parameters we want)
    Y = measurement (noisy observations about X)
  - At each time step t, state changes from X_{t-1} to X_t, and get new observation Y_t
- Our goal: Estimate distribution of state X_t, given...
  - All observations so far
  - Knowledge about dynamics of state transitions

2015-10-18(b)

- Prediction: What is next state of object, given past measurements?

$$P(X_t | Y_0 = y_0 \dots Y_{t-1} = y_{t-1})$$

- Correction: Compute updated estimate of the state from prediction & measurements

$$P(X_t | Y_0 = y_0 \dots Y_{t-1} = y_{t-1}, Y_t = y_t)$$

→ after the measurement at time t

- Tracking: Process of propagating this posterior distribution of state given measurements across time

- To simplify:
  - Markovian assumption: Only immediate past matters

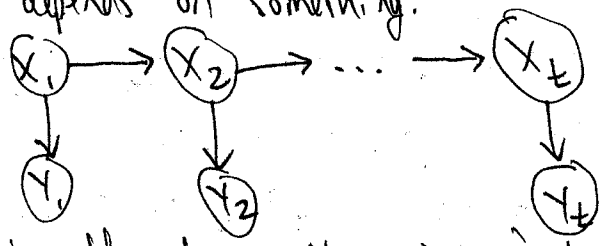
$$P(X_t | X_0 \dots X_{t-1}) = P(X_t | X_{t-1}) \rightarrow \text{Dynamics model}$$

  - Measurements depend only on current state

$$P(Y_t | X_0, Y_0, \dots X_{t-1}, Y_{t-1}, X_t) = P(Y_t | X_t) \rightarrow \text{Observation model}$$

    - this can be shaky, though, if noise is present that depends on something.

- Graphically:



  - For those in the know this is just a Hidden Markov Model! (Machine Learning folks recognize this typically).

- Tracking as induction
  - Base case: Assume some initial prior, predicting state w/o anything else — $P(X_0)$
  - At first frame, 'correct' this with given $Y_0 = y_0$.
  - Given corrected frame t estimate: Predict $t+1$, correct $t+1$
- Prediction: given $P(X_{t-1} | y_0 \dots y_{t-1})$, guess $P(X_t | y_0 \dots y_{t-1})$
  - See derivation on p11
  - guess then = $\int P(X_t | X_{t-1}) \overbrace{P(X_{t-1} | y_0 \dots y_{t-1})}^{= \text{given}} dX_{t-1}$
- Correction: given $P(X_t | y_0 \dots y_{t-1})$ predicted value, and y_t, compute $P(X_t | y_0 \dots y_t) = \frac{P(y_t | X_t) P(X_t | y_0 \dots y_{t-1})}{\int P(y_t | X_t) P(X_t | y_0 \dots y_{t-1}) dX_t}$

7B-L2

- The Kalman filter
  - Linear dynamics model
    - Dynamics model: State undergoes linear transformation plus Gaussian noise

$$x_t \sim N(D_t x_{t-1}, \Sigma_{d_t})$$

We write D_t but D is generally constant

    - Observation model: $y_t \sim N(M_t x_t, \Sigma_{m_t})$
  - Example: Constant velocity

$$x_t = \begin{bmatrix} p_t \\ v_t \end{bmatrix} \quad p_t = p_{t-1} + (\Delta t) v_{t-1} + \varepsilon \text{ (noise)}$$
$$v_t = v_{t-1} + \varepsilon_2$$

$$x_t = D_t x_{t-1} + \text{noise} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{t-1} \\ v_{t-1} \end{bmatrix} + \text{noise}$$

  - Measurement might be position only

2015-10-18c

- Or, constant acceleration, $x_t = \begin{bmatrix} p_t \\ v_t \\ a_t \end{bmatrix}$

- Kalman filters are built on Gaussian noise/distributions
- "shift, expand, measure, correct, shrink"
  - new information narrows your distribution
- What Kalman gets you: you may trade off between predicted & measured values, based on their respective uncertainties (variances)
- I'm really not taking many notes here... There is a lot else in slides & lectures.
- This is the first I've seen Kalman Filters phrased thus, but it's helpful.
- Why your covariance is <u>lower</u> than your measurement's variance: Because new information can only decrease your covariance!
- Cons of Kalman filter:
  - unimodal distribution, only single hypothesis
  - restricted class of motions defined by <u>linear</u> model
    - See extended Kalman filter, though. It linearizes using Jacobian.
- So: What do we do if not Gaussian, or not even unimodal?

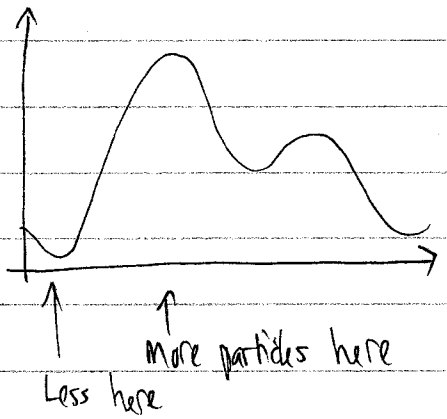
$P(A|B)$ = probability of A under condition of B

$$= \frac{P(A \cap B)}{P(B)}$$

2016-10-20

7C-4

- Bayes filters
  - N.B. for later notes: Here (in particle filtering) we write measurements as z_t, not y_t.
  - Basic idea w/ particle filters.



Density is both where particles are, and their weight.

$p(x=x_0)$ is probability of drawing x with value x_0.

Goal: $p(x_t \in X_t) \approx p(x_t | z_{1:t})$ with equality as $n \to \infty$

- Given all of our observations we want them to be able to give us that density

- Given:
  - Prior probability of system state, $p(x)$
  - Action (dynamical system) model, $p(x_t | u_{t-1}, x_{t-1})$
  - Sensor model, $p(z|x)$
  - Stream of observations z & action data u (or input),

$$data_t = \{u_1, z_2, \ldots u_{t-1}, z_t\}.$$

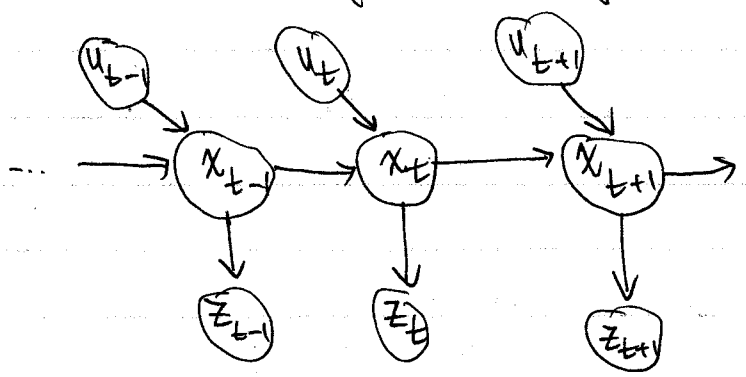
- Sensor model is not, "what is the likely state x, given my measurement z". It is, "what is my likely measurement z, given state x". This matters!

- Wanted:
  - Estimate of state X at time t
  - Posterior of the state, or belief:

$$Bel(x) = P(x_t | u_1, z_2, \ldots u_{t-1}, z_t)$$

2015-10-20(b)

-For machine learning folks, a graphical representation:



Assumes static world & independent noise

$p(z_t | x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t | x)$ - Measurement is only current state, aka sensor independence

$p(x_t | x_{1:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t | x_{t-1}, u_t)$

- Probability of next state is only a function of the last state and the input (Markovian property).

-Recall Bayes Rule:

$$p(x|z) = \frac{p(z|x)\,p(x)}{p(z)}$$

(prior before measurement; $p(z)$ ← once we have our measurement, this probability is done, it won't change.)

$$= \eta\, p(z|x)\,p(x)$$

$$\propto p(z|x)\,p(x)$$

- Since it must sum to 1 we can handle it just with normalization constants.

-So...

$$Bel(x_t) = P(x_t | u_1, z_2 \ldots u_{t-1}, z_t)$$

$$\text{Bayes} = \underbrace{\eta\, P(z_t | x_t, u_1, z_2, u_{t-1})}_{\text{Likelihood}} \underbrace{P(x_t | u_1, z_2, \ldots u_{t-1})}_{\text{Prior}}$$

This reduces because only the current state matters.

Sensor independence: $= \eta \, P(z_t|x_t) \, P(x_t|u_1, z_2, \ldots u_{t-1})$

Total probability of prior: (note this is prior ↑)

$= \eta \, P(z_t|x_t) \int P(x_t|u_1 z_1 \ldots u_{t-1} x_{t-1}) \, P(x_{t-1}|u_1, z_1, \ldots u_{t-1}) \, dx_{t-1}$

↑ Sum of all possible x_{t-1}

What is probability of going to x_t, given prior state, all prior inputs & measurements? ↑

- But, because of Markovian rule most of this doesn't matter: and reduces to:

Markov: $= \eta \, P(z_t|x_t) \int P(x_t|u_{t-1}, x_{t-1}) \underbrace{P(x_{t-1}|u_1, z_1, \ldots u_{t-1})}_{\text{This is just } Bel(x_{t-1})!} dx_{t-1}$

Or: $\boxed{Bel(x_t) = \eta \, P(z_t|x_t) \int P(x_t|u_{t-1}, x_{t-1}) \, Bel(x_{t-1}) \, dx_{t-1}}$

So, this is clearly an inductive formula.

$Bel(x_{t-1})$, when multiplied by $P(x_t \ldots)$ in integral, is just our prediction before measuring.

(Similar to Kalman, really.)

Then, $P(z_t|x_t)$ is just: How likely was this measurement?

7C-L2 2015-10-21

- Particle filters
  - We base this on the Bayes Filters of last chapter
  - Slide 19 gives algorithm.

2015-10-21(b)

7C-L3
- Particle filters for localization
- This can be multiple hypothesis (lecture 3 talks about this a little)
- Review the slides & lectures here!

7C-L4
- Particle filters for real
  - What is the state x? What's it mean?
  - What is the measurement? How's it relate to state?
  - Where do your dynamics come from?
- Paper establishing much of this:
  M. Isard & A. Blake, 1998. CONDENSATION - Conditional Density Propagation for Visual Tracking
- Note that the "particles" here are in a 6-dimensional space if our state is an affine transformation (like the example of tracking the girl's head as she moves in the frame)
  - This was not clear to me before.
- See: Head Tracking with Contour Models (Zhihong Zeng, et al. 2002)

7D-L1
- Tracking considerations
  - Mean-shift algorithm is fairly simple, and gets you to the mode/center-of-mass of a PDF easily.

Kernel-Based Object Tracking,
Comaniciu, Ramesh, Meer

- Mean-shift object tracking... is similar?
  - One can use gradient-descent to find the mode
  - given a representation in a feature space of some kind & similarity function
- With a similarity function, you can use this as a sensor model for particle filtering (looking for a "best" is more like Kalman Filters & single hypothesis)

- Tracking issues
  - Initialization (How do you start it?)
    - Manual
    - Background subtraction
    - Detection (e.g. a particular car type)
  - How do you obtain observation & dynamics model?
    - Dynamics model:
      - From real data (very hard - if you can track already, then why do you need it?)
      - From "clean data" (e.g. black background
      - Specify using domain knowledge
    - Generative observation model - some ground truth needed (e.g. the top-down map the robot uses)
  - Prediction vs. correction
    - If dynamics model is too strong - ignores data.
    - Observation model too strong - tracking reduces to repeated detection

2015-10-22

- Data association
  - How do we know which measurements associate with which tracks?
  - In reality, we may not know, and reality has multiple objects or clutter (useless measurements)
  - Simple strategy: pay attention to closest measurement to prediction. (Not robust to clutter)
  - More sophisticated: Keep track of multiple state/observation hypotheses.
    Each particle is a hypothesis about current state.
- Drift - accumulated error in dynamical model, observation model, & data association
  - You must update your model - ~~the hypothesis~~ your object may change across time somewhat
  - But if you adapt too quickly, it may start to track the wrong thing
  - So, it requires tuning. Both approaches have problems.

2015-10-25

8A-L1

- Introduction to recognition
  - All that we did so far was "semantic-free" - it didn't include anything about identifying/recognizing the objects
  - We cover only a little of this, because so much of this (object recognition particularly) now fits under machine learning, applied to pixel intensity patterns
- May involve:
  - Verification: Is this region a lamp?
  - Identification in class: Are there people in this region?
  - Identification of specific thing
- Instance-level recognition (John's car) vs generic categorization (a car)
- Task: "Given a (small) number of training images of a category, recognize a-priori unknown instances of that category and assign the correct category label."
- But, any object has multiple possible categories; which one do we want?
- To be meaningful for humans, usually we pick highest level at which category members have similar perceived shape. (You can visualize or draw "German shepherd". You can likewise for "dog". For "mammal", though, you likely have no good visualization.)
- Aaron Bobick's work: "Natural Object Categorization"

2015-10-25(b)

- We also use functional categories (e.g. chairs as 'something you sit on'), ad-hoc categories (e.g. office supplies)
- Normally though we categorize visually
- "Basic Level Categories in human categorization" (Rosch 76, Lakoff 87)
- Why is this difficult?
  - Robustness to illumination, object pose, clutter, occlusions, intra-class appearance, viewpoint
  - Context is very important sometimes.
  - Complexity (# of pixels, # of images, # of object categories, # of dimensions)
    - About half of the cerebral cortex in primates is for processing vision!
- What's easy?
  - Reading license plates, ZIP codes, fingerprints, face detection, flat textured objects (CD covers, book covers)
- GoogleNet 2014?
- Next few lectures:
  - Generative vs. discriminative methods
  - Activity recognition

8B-L1

- Classification: Generative models
  - Supervised: Given collection of labeled examples, create a function that will predict labels of new examples.
  - How good it is:
    - What mistakes does it make? How expensive are mistakes?
  - We want to minimize expected misclassification
- General strategies:
  - Generative: Build representative probability model from training data;
  - Discriminative: Provide it with examples of category A, and of category... not-A. Find a way to discriminate between the two.
- We focus on generative here.
  - Notation: $(4 \to 9)$ - object is a 4, we call it 9.
  - We assume cost of $(X \to X)$ is zero.
  - $L(4 \to 9)$ = loss (cost) of classifying 4 as 9
  - Risk of classifier strategy S is expected loss:

$$R(S) = \Pr(4 \to 9 \mid \text{using } S)\, L(4 \to 9) + \Pr(9 \to 4 \mid \text{using } S)\, L(9 \to 4)$$

  - "Best" decision boundary: either choice of label yields same expected loss. (If it were different we'd just move the boundary.)
    That is: $P(\text{class is } 9 \mid x)\, L(9 \to 4) = P(\text{class is } 4 \mid x)\, L(4 \to 9)$
- And how do we get these probabilities?
  Use: $P(A|B) = \dfrac{P(B|A)\,P(A)}{P(B)}$, $P(A|B) \propto P(B|A)P(A)$

2015-10-26

- Apply Bayes' rule... see slide 13ish.
- But where does the prior (i.e. $P(A)$) come from?
- We want:
  $P(skin|x) > P(\neg skin|x)$ - This is useful for detection.
  $\equiv P(x|skin)P(skin) > P(x|\neg skin)P(\neg skin)$ - Use priors.
- Just re-weight if costs are different.
- But we don't have prior, $P(skin)$. However, we can estimate it from training data.
  That same training data can give $P(x|skin)$ and $P(x|\neg skin)$.
- Look up CAMSHIFT (Gary Bradski)

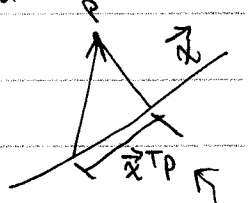
- We've been doing two-class models. To generalize:
  - Given measurement x and set of classes c_i,
    find $c^* = \underset{c}{\text{argmax}}\ p(c|x) = \underset{c}{\text{argmax}}\ p(c)\,p(x|c)$
- If x continuous, need <u>likelihood density model</u> of $p(x|c)$?
- Why have a continuous generative model though?
  - If you have lots of data, maybe use histogram or KNN.
  - The point of using a parametrized model is <u>not</u> to need lots of data.

- This method predates a lot of modern computers, as it tries to substitute some pure mathematical reason in rather than using a lot of CPU cycles.

- Why generative models:
  - Firm probabilistic grounding
  - Allows use of prior knowledge!
  - Parametric ~~likelihood~~ modeling of likelihood permits using small number of examples
  - New classes don't perturb previous models?
  - Can be used to generate samples
- Why not:
  - Getting priors is sort of cheating...
  - Why model all those non-category points?
  - Lots of data doesn't help!

8B-L2

- Principal Component Analysis (PCA)
  - This is useful in general but we apply it here w/ generative models
  - N.B. Generative models work better in lower-dimensional spaces
  - PCA: All about directions in a Feature space along which points have greatest variance
  - First PC (principal component) is direction of max variance.
  - Subsequent PCs are orthogonal to the last and describe max residual variance.
  - Max variance should jog your memory about fitting a line. (See slides 4 & 5)
  - Minimizing sum-of-squares is just $\sum_i (\vec{x}^T P_i)^2$ maximized
    - $\vec{x}^T$ = line, B^T = points, ~~[illegible]~~,
    - That is: Maximizing sum of squares of projections on that line.



adriancarsiotis@gmail.com

2015-10-266

- Thus: $\max(x^T B^T B x)$ under $x^T x = 1$
  - Maximize $E = x^T M x$ w/ $x^T x = 1$ $(M = B^T B)$
    Let $E' = x^T M x + \lambda(1 - x^T x)$
    → Lagrange multiplier, λ.
    If we solve right then $1 - x^T x = 0$, so $E' = E$ and λ doesn't matter.
    $\partial/\partial x\, E' = 2Mx + 2\lambda x = 0$ (to minimize)
    → $Mx = \lambda x$
    → x is an eigenvector of $B^T B$ (M), λ is eigenvalue

$x^T = [\text{Line}]$

$B^T = \begin{bmatrix} \text{P}_1 & \text{P}_2 & \text{P}_3 & \cdots \end{bmatrix}$

$B = \begin{bmatrix} \text{Point } 1 \\ \text{Point } 2 \\ \cdots \end{bmatrix}$

$X = \begin{bmatrix} \text{L} \\ \text{i} \\ \text{n} \\ \text{e} \end{bmatrix}$

- Another interpretation: $B^T B = \begin{pmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i y_i \\ \sum x_i y_i & \sum y_i^2 \end{pmatrix}$ if around origin

  which is just <u>covariance matrix.</u>
  Principal components then are: orthogonal directions of the covariance matrix.
- Yet another: $B^T B = \sum x x^T$ if about origin
  $= \sum (x - \bar{x})(x - \bar{x})^T$ otherwise (outer product)
- Real symmetric $N \times N$ matrices generally have N distinct eigenvectors
- Why: Dimensionality reduction
- Algebraically: Suppose you've N-dimensional data points

2015 10 27

$$\text{var}(v) = \sum_x \| (x - \bar{x})^T \cdot v \| = v^T A v$$

where $A = \sum (x - \bar{x})(x - \bar{x})^T$ ← Outer product

- Eigenvector w/ largest eigenvalue λ captures <u>most</u> variation (smaller eigenvalues → less variation)

- N.B. PCA isn't as useful when the data may have non-orthogonal components you want.
Something like ICA, Independent Component Analysis.

- Think of a face image, 100×100 pixels.
Seen as a vector, that's 10,000 dimensions.
  - But very few 10,000-dimension vectors are 'valid' faces.
  - We want a lower-dimensional linear subspace to best explain variations.

- PCA, more formally:
  - m data points, $x_1 \dots x_m$, in $\mathbb{R}^d$, where d is big.
  - We want directions in $\mathbb{R}^d$ that capture most variation of x_i. Coefs would be:
$u(x_i) = u^T(x_i - \mu)$ where μ = mean of points
  - So, what unit vector $u \in \mathbb{R}^d$ captures most variance?

$$\text{var}(u) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{u^T(x_i-\mu)}_{\text{Projection of point}} \left(u^T(x_i-\mu)\right)^T = u^T \underbrace{\left[\frac{1}{m}\sum_{i=1}^{m}(x_i-\mu)(x_i-\mu)^T\right]}_{\text{Covariance matrix } (=\Sigma\ ?)} u$$

? RHS is dot product, LHS is ???

$$\text{var}(u) = u^T \Sigma u$$

  - Since u is a unit vector that can be expressed in terms of some linear sum of the eigenvectors of Σ, then direction of u that maximizes variance is the eigenvector associated with the largest eigenvalue of Σ.
  - Note that $\Sigma v = \lambda v$ for eigenvector v, eigenvalue λ of Σ.
Hence we want the largest λ here to maximize var(u).

Why? I think it's because Σ is symmetric positive semi-definite, thus its eigendecomposition forms an orthogonal basis, thus it spans $\mathbb{R}^d$ and $u \in \mathbb{R}^d$.

2015-10-27(b)

- Further the top k orthogonal directions that capture the most variance of the data = the k eigenvectors w/ the k largest eigenvalues.
- A trick exists for $d >>> M$
  - Let Φ_i = d-vector with mean subtracted.

$$C = 1/M \sum \Phi_i \Phi_i^T = AA^T$$

where A is all Φ_i stacked - that is, $\left[\Phi_1, \Phi_2 \ldots \Phi_n\right]_{d \times M}$

C is then rather huge, d x d, which is often impractical for eigendecomposition.

- Now consider A^TA: This is just MxM - much smaller.

Suppose v_i is an eigenvector of A^TA, then by definition, $A^TAv_i = \lambda v_i$ and then premultiply by A:

$AA^TAv_i = A\lambda v_i = \lambda Av_i$ (λ is scalar)

$C(Av_i) = \lambda(Av_i)$ which means that Av_i is an eigenvector of C - which is what we wanted!

$C = AA^T$

- If $M > d$, there'd be d eigenvectors (i.e. if we had more data points/images than dimensions)
- But here, $M <<< d$. Intuition says M eigenvectors here. But, intuition is wrong: subtracting out the mean removes a degree of freedom. So, it is <u>M-1</u> eigenvectors.

- Applied to faces, we get: Eigenfaces (Turk & Pentland, 1991)
  - Assume most spaces lie on low-dimensional subspace w/ first k directions of max variance ($k <<< d$)

- Use PCA to find vectors ("eigenfaces") spanning that subspace.
- Now: Represent all face images as linear combinations of faces, and store them as coefficients!
- See slide 28 for example → w coefs
- Then merely take a dot product to reconstruct
- To actually get coefficients:

$$x \rightarrow \left[u_1^T(x-\mu), \ldots u_k^T(x-\mu)\right] = \underbrace{\left[w_1, \ldots w_k\right]}_{\text{face-space coefficients}}$$

(u_i: eigenvector, μ: mean)

- But, we want this for recognition, not reconstruction. To 'recognize' a face, just project onto subspace, and compare these. - find 'closest' training face.
  - Another cool feature: Check $x - \hat{x}$ (reconstruction error) to see if it's really a face...
- That's why this is a generative model - classify as the closest training face in k-dimensional space

- Limitations: (PCA of global structure)
  - Not robust to misalignment & background clutter
  - Dir. of max variance isn't always good for classification!
    - See: Fisher analysis, linear discriminant analysis
- Eigen-gaits? Eigen-body shapes? PCA can sort of do this.
- This method is linear and everything's nice and orthogonal. This is also a liability - it can't capture if points are in a curved space/manifold/line that is lower-dimensional

See: non-linear kernel PCA

2015-10-29

SB-L3

- Appearance-based tracking
  - Some of these PCA methods & eigenfaces can be used for detection
  - Another task is related: Tracking
  - Basic idea: Find the low-dimensional representation, and track wherever you can reconstruct the best
- Conventional tracking: Optical flow
  Patch tracking w/ particles
  Solve optimization problem...
  - Add invariance to cope w/ pose, lighting, view angle variation
- Problem: Object appearance & environment always changing.

- See: Eigentracking (Black & Jepson 96)
  - Key insight (over eigenfaces): Separate geometry from appearance with deformable model.
  - This needs non-linear optimization however...
  - Needs fixed basis set?
- Active Contour (Isard & Blake) - Use particle filter, propagate uncertainty over time

- Incremental Visual Learning (Ross, Lim, Lin, Yang, 2007)
  - Aims: Not single image based, updates model, works w/ moving camera
  - Challenge: Pose variation, occlusion, illum. change, drifts

1. Adaptive visual tracker
   - ~~Draw samples from~~ Particle tracking on deformation
2. Subspace-based tracking
   - Track like with eigenfaces
3. Incremental subspace update: Doesn't build model before tracking, handles variation in lighting/pose/expression

- Location at time t: L_t — similarity transform = $(x_t, y_t, \theta_t, s_t)$; Affine (6 DOF)
- Current observation: F_t
- Goal: Predict target location L_t based on L_{t-1} (location in last frame)

$$p(L_t | F_t, L_{t-1}) \propto p(F_t | L_t) \cdot p(L_t | L_{t-1})$$

$p(F_t | L_t)$: observation model (eigenbasis w/ approximation)

$p(L_t | L_{t-1})$: dynamics model, Brownian motion (no velocity)

- Observation model is higher probability if eigenbasis can reconstruct well, lower otherwise
- Dynamics, $p(L_t | L_{t-1})$
  - $p(L_1 | L_0) = N(x_1; x_0, \sigma_x^2) N(y_1; y_0, \sigma_y^2)$ ---
  - Or: Each parameter is independently Gaussian distributed, with mean at the last result. That is, it's less likely that the parameter drifts further, and less likely the further you go.

2015-10-29b

- Observation model, $p(F_t | L_t)$
  - Probabilistic PCA to model image observation
  - Given location L_t assume observed frame was generated from eigenbasis

$$p(z|B) = N(z; \mu, BB^T + \varepsilon I)$$

- z: datum
- B: eigenbasis
- μ: mean
- εI: additive noise
- $p(z|B)$: probability of observing z given eigenbasis B

As $\varepsilon \to 0$, this is mostly dependent on <u>distance</u> between z & B.

$$p(z|B) \propto \exp\left(-\left\| (z-\mu) - \underbrace{BB^T(z-\mu)}_{\text{Reconstruction}} \right\|^2\right)$$

(see slide 21 for why)

  - This is just the distance to the eigenspace itself. Or: This is how much signal <u>can't</u> be reconstructed with eigenbasis. (residual?)

"Incremental Subspace Update"

- Now, allow our observation model to change over time!
  - We compute a new eigenbasis at each step. Find B_t from B_{t-1} & observation w_{t-1}, then use B_t in $p(F_t | L_t)$.
  - They use R-SVD algorithm (Golub & Van Loan 96) and sequential Karhunen-Loeve algorithm. In essence, this computes a running mean, but with decay (older values matter less)

- Total form:

1. Optional: Construct initial eigenbasis (if needed for initial detection)
2. Choose initial location L_0
3. Generate possible locations: $p(L_t | L_{t-1})$
4. Evaluate " " $p(F_t | L_{t-1})$
5. Select most likely by Bayes': $p(L_t | F_t, L_{t-1})$
6. Update eigenbasis w/ R-SVD

- Recent work: Handling occlusion. (The above can't really track past this.)
  - One method estimates "weight mask" which gives the 'confidence' in pixels, and estimates likelihood that the pixel is occluded

2015-11-09

8C-4

- Discriminative classifiers
  - Rather than modeling classes (as in generative), probabilistically model the <u>decision</u>.
- Generative approaches were among first used in pattern recognition, but generative models have liabilities in the "modern world"
  - Many signals are high-dimensional, and representing complete density of class is "data-hard" (grows exponentially with dimension)
  - Also, we don't care about modeling classes - we just want to make the right decision.
    So model the hard cases, i.e. the <u>boundaries</u>.
  - Also, what does modelling non-classes even mean? (e.g. if we want to recognize skin, how do we model the class of non-skin?)
  - We may not know what features actually discriminate between classes.
    - We may then have to figure out what features here are salient.

- 'Classification' & 'recognition' differ but the notes and slides use them basically as the same.

- Assumptions for discriminative classification:
  - Fixed number of known classes!
  - Ample training examples in each class
  - Equal cost of mistakes (what matters is getting label right)
    - Relative cost is irrelevant to us
  - We do not know a priori what features are diagnostic of the class label - but we must construct a representation of the instance

- Two main parts of building:
  - Build an object model (a representation)
    - Describe training instances (here, images)
  - Learn/train a classifier
- Testing: - Generate new candidates, and score them

- For image-based detection:
  - Often we use 'windowing methods' where we refer to specific regions in image
  - One very simple holistic description: greyscale/color histogram
    - Or: Vector of pixel intensities (as we did with PCA)
    - But this is not robust to even tiny shifts
    - Greyscale/color histogram isn't robust to changes in illumination or intra-class variation
  - But consider what we had to do for tracking & motion detection - and interest points and gradients and Harris corners

2015-11-09b

- Those are much more robust, especially paired w/ SIFT.
- Think of dividing image into quadrants and doing a similar approach in each quadrant. That resolves many issues.

- Now: Given lots of these, how to learn/train classifier?
  - Binary classifier - just two classes (e.g. car/not-a-car)
  - Slide 25 has many ways of building classifier:
    - Nearest neighbor
    - SVMs (Support Vector Machines)
    - Boosting

    ] We talk about these

    - Neural networks
    - Random Forests

    ] And ignore these

  - Ideally, someone hands you a database of examples of your class, and examples of not-your-class
    - To train you often must put different-sized windows all across image (which is very data-intensive)

- Training:
  - Obtain training data
  - Define features
  - Train classifier

- Given new images:
  - Slide window
  - Score by classifier

- Discriminative classifiers: Find division (surface) in feature space that separates classes.
  - Nearest neighbor (slide 33)
    - Have a Voronoi partitioning of feature space w/ positive & negative data points (in class/not in class)
    - Problem with this: Doesn't work very well, & very data-intensive

Note that Voronoi partitioning divides space up by nearest neighbor.

- K-Nearest Neighbor
  - Instead of finding just one neighbor, find nearest k points from training data.
    - These 'vote' to classify (e.g. 5 NN, with 3 negatives & 2 positives → not in class)
  - This works better; it has a notion of consensus.
    - Still has some computational issues

So generally one sets k odd to always have a majority.

3/4-12

- Boosting & Face Detection
  - Instructor considers this really cool & pivotal!
- Training method w/ boosting:
  - Initially, weight each training round equally
  - In each boosting round (it's iterative):
    - Find weak learner that gives lowest weighted training error
    - The training examples that the current weak learner misclassifies: raise their weights
  - But what is a weak learner? It's just a function that partitions your space. It doesn't always give a right answer, but gives something.
  - The final classifier is simply a linear combination of all weak learners (weight of each ∝ accuracy).
- Example at slide 8 & 2nd video...
- Exact re-weighting & combining formulas depend on particular boosting scheme (e.g. AdaBoost)
- Very important paper: "Rapid object detection using a boosted cascade of simple features" (P. Viola & M. Jones, CVPR 2001)

weak learner = weak classifier

Viola-Jones face detector

2015-11-09 c

- Good ideas from Viola-Jones:
  - Represent brightness patterns w/ efficiently computable "rectangular" features in window of interest
    - Sort of a large-size gradient, applied to different image parts.
    - It only looks at difference in average intensity.
  - It uses an integral image: Value at (x,y) = sum of pixels above & to left of (x,y)

slide 21

slide 23

  - Why: You can very efficiently compute sums of rectangular image regions! (For any size rectangle, you need only 3 additions - sum = A - B - C + D)
  - And: No need to scale images! Just scale the features directly (cost is exactly the same)

Hence, the use of integral images.

- The training for Viola-Jones used 180,000+ possible features in each 24x24 window. It considered all possible filter parameters - position, scale, type.
  - Then AdaBoost was used to find the subset of those features that are informative & form a classifier.
- Next Viola-Jones ideas:
  - Choose discriminative features to be weak classifiers/learners
  - Use boosted combination as final classifier.
  - Form a cascade of such classifiers, rejecting clear negatives quickly

★

    - Even if the filters are fast, a whole lot of possible windows to search exist. How do we make detection efficient?

Key with Viola-Jones: Slow training, very fast detection
- Really useful idea is cascade filter

- Insight: Almost everywhere is <u>not</u> a face. So: detect non-faces quickly, be very certain it's not a face, & move on.
- The cascade puts really low <u>false negative</u> rates early.
  - At each following stage, feed it <u>false positives</u> from last stage ("difficult negatives") - and use these to train this stage!
  - Then repeat to form a cascade where each stage deals with more & more difficult negatives...

- Boosting advantages:
  - Combines feature selection & classification
  - Very flexible w/ what weak learners & boosting scheme you use
  - Training is <u>linear</u> in # of training examples
  - Testing is very fast
  - Easy to implement
- Disadvantages:
  - Needs <u>lots</u> of training examples
  - Doesn't work as well as SVM & random forests sometimes, esp. for many-class (not binary-class like here)

8C-L3
- Support Vector Machines
- First: <u>Linear classifiers</u>. They're just linear separators that divide two classes. SVMs are a higher-dimensional form of this.
- In $\mathbb{R}^2$: Let $w = \begin{bmatrix} p \\ q \end{bmatrix}$, $x = \begin{bmatrix} x \\ y \end{bmatrix}$, $px + qy + b = 0$
  $\Leftrightarrow$ $w \cdot x + b = 0$

2015-11-09d

- Distance from point (x_0, y_0) to line:

$$D = \frac{|px_0 + qy_0 + b|}{\sqrt{p^2+q^2}} = \frac{w^T x + b}{|w|}$$

- Linear classifiers:
  - We want a function to separate 'positive' and 'negative' x_i:
    - x_i 'positive': $x_i \cdot w + b \geq 0$ — but this might have multiple solutions
    - x_i 'negative': $x_i \cdot w + b < 0$
- SVMs need 'optimal separating line': Maximize <u>margin</u> between positive & negative training examples.
  - Let our line be $wx + b = 0$
  - Positive margin: $wx + b = 1$
  - Negative margin: $wx + b = -1$

  ] Note that we can scale w & b however we want since formula of line equals 0

  - So, for 'positive' x_i, $x_i \cdot w + b \geq 1$ (& let $y_i = 1$)
    'negative' x_i, $x_i \cdot w + b \leq -1$ (& let $y_i = -1$)
  - Certain x_i are special because they define our <u>margin</u>. Those points are our... <u>support vectors</u>.
  - Point-line distance is still:

$$\frac{|x_i \cdot w + b|}{\|w\|}$$

  - For support vectors, $w^T x + b = \pm 1$
  - So... Margin $M = \left| \frac{1}{\|w\|} - \frac{-1}{\|w\|} \right| = \frac{2}{\|w\|}$

M is distance between support vectors.

- We want to <u>maximize</u> $M = 2/\|w\|$ such that we correctly classify all points. (That's this constraint)

Slide 6,7

Recall that $y_i = 1$ for 'positive' x_i,
$y_i = -1$ for 'negative' x_i

- This is then a quadratic optimization problem:
  - Minimize $\frac{1}{2} w^T w$ ($\|w\| = w^T w$, and minimizing this maximizes $\frac{2}{\|w\|}$)
  - Subject to $y_i(x_i \cdot w + b) \geq 1$
- Solution (abbreviated): $w = \sum_i \alpha_i y_i x_i$
  - α_i = learned weight, non-zero only at support vectors
  - x_i = support vectors
  - $b = y_i - w \cdot x_i$ for any support vector
  - $w \cdot x + b = \sum_i \alpha_i y_i x_i \cdot x + b$
  - Classification function: $f(x) = \text{sign}(w \cdot x + b) = \text{sign}\left(\sum_i \alpha_i x_i \cdot x + b\right)$ (dot product only)
  - $f(x) < 0$ = negative classification
  - $f(x) > 0$ = positive classification
  - Note that $f(x)$ depends only on dot product
  - We choose support vectors simply by what α_i we set $\neq 0$
- What if...
  - Not 2D? (doesn't matter. The math is agnostic.)
  - Not linearly separable?
  - More than 2 categories?

slide 19

- If not linearly separable:
  - Map to a higher-dimensional space. (e.g. $x \to x^2$)
  - Say that $\phi : x \to \varphi(x)$ where it moves to higher-dimensional space whose features are separable.
  - Define $K(x_i, x_j) = x_i \cdot x_j = x_i^T x_j$ which becomes $K'(x_i, x_j) \Rightarrow \varphi(x_i)^T \varphi(x_j)$ if we map with ϕ
  - K stands for kernel — a similarity function corresponding to an inner product in <u>expanded</u> feature space.
    (and some other constraints I don't get)

N.B.: We don't <u>care</u> what the high-dimensional space is that ϕ maps to. We just must be able to show that K = dot product in some space.

2015-11-09e

- Example: 2D vectors $x = [x_1, x_2]$

Let $K(x_i, x_j) = (1 + x_i^T x_j)^2$

Need to show: $K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$

for some space.

Multiply K out:

$$K(x_i, x_j) = 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2}$$

$$= [1 \quad x_{i1}^2 \quad (\sqrt{2} x_{i1} x_{i2}) \quad x_{i2}^2 \quad \sqrt{2} x_{i1} \quad \sqrt{2} x_{i2}]^T$$
$$[1 \quad x_{j1}^2 \quad (\sqrt{2} x_{j1} x_{j2}) \quad x_{j2}^2 \quad \sqrt{2} x_{j1} \quad \sqrt{2} x_{j2}]$$

$$= \varphi(x_i)^T \varphi(x_j)$$

where $\varphi(x) = [1 \quad x_1^2 \quad \sqrt{2} x_1 x_2 \quad x_2^2 \quad \sqrt{2} x_1 \quad \sqrt{2} x_2]$

We mapped a 2D space to 6 dimensions (our dot product is defined there). We don't actually care about our space - we just use the kernel trick with K:

$$\sum_i \alpha_i y_i (x_i^T x) + b \rightarrow \sum_i \alpha_i y_i \overbrace{K(x_i, x)} + b$$

- What kernels are good?
  - Simplest: Linear, $K(x_i, x_j) = x_i^T x_j$ (same dimensions)
  - Gaussian RBF (radial basis function): $K(x_i, x_j) = \exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right)$
  - Dot product in infinite dimensions:

$$K(x, x') = \sum_{j=0}^{\infty} \frac{(x^T x')^j}{j!} \exp\left(\frac{-1}{2}\|x\|_2^2\right) \exp\left(\frac{-1}{2}\|x'\|_2^2\right)$$

(ignoring $2\sigma^2$)

Relates to e^x infinite series.

See: "classification using intersection kernel support vector machines is efficient" (Berg & Malik, CVPR 2008)

- Another: Histogram intersection, $K(x_i, x_j) = \sum_k \min(x_i(k), x_j(k))$
  - Number of dimensions: "Large"
- Many others exist; histogram intersection kernel is very simple but very effective

2015-11-11

- SVMs for recognition: Training
  - Define representation
  - Choose kernel function (he'll go into how later)
  - Compute pairwise kernel values between labeled examples
  - Use this 'kernel matrix' to solve for SVM support vectors & weights
- This doesn't give you a line, but some other surface in your space as your boundary.
  - Basically, you use that kernel to 'bend' your boundary how you need.
- To use this to predict:
  - Compute kernel values between new input & support vectors
  - Apply weights
  - Check sign of output
- Slide 32: Gender classification using this
  - Used 1044 males, 713 females, for training. Images were shrunk to just 21x12 pixels.
  - Kernel was Gaussian RBF
  - The support vectors are then 'support faces': The faces that are closest to the boundary and thus hardest to label
  - SVM actually outperformed humans here!

2015-11-11(b)

- Next question: What if you have ≥ 2 classes?
  - SVMs by themselves are just binary classifiers.
  - One way is <u>one vs. all</u>
    - Training: SVM for each class vs. the rest
    - Testing: Apply each SVM to new input, and choose SVM with highest decision value (your function tells you how 'far' it is from the boundary, not just a yes/no)
  - Another is <u>one vs. one</u>
    - Training: SVM for each <u>pair</u> of classes
    - Testing: Each SVM 'votes' for a class to be assigned
  - Lecturer says: Neither way is satisfactory but most use one vs. all.
- Pros to SVM
  - Many publicly-available libs (e.g. libsvm)
  - Kernel-based frameworks are very flexible
  - While training is very lengthy (maybe), the result is very compact and testing is very fast!
  - Works very well in practice, even w/ small training samples
- Cons:
  - No 'direct' multi-class SVM
  - Choosing the right kernel is hard. Not much theory is here. Most folk use linear, Gaussian RBF, or histogram intersection.

- The flip side of many packages/libraries being available is that no one writes their own and then they're at the whim of what parameters the library exposes - and lose some intuition about what's going on.
- Training is very compute-heavy. It can be a lot of data and a lot of computation. (Need matrix of kernel values for every pair of examples. $O(N^2)$...)

- Next topic is random forests. Kinect uses this for detecting limbs and such, and you actually can implement it yourself.
- He stops here because these go more into machine learning, merely the computer vision specializations that work over larger feature vectors.

8C-L4

- Bag of Visual Words
  - Used a lot in video, somewhat in static images
  - Think back to SIFT...
  - Every patch/region has a descriptor in some high-dimensional feature space (e.g. SIFT)
  - Nearby points in feature space indicates similar local content.
  - But, how would we efficiently search potentially thousands/millions of matches?

2015-11-11c

- So: Build an <u>index</u> of all <u>images</u> in which some feature occurs.
  Maybe if we find an image with a lot of the same features, it's related to another.
- The index on a page isn't really ordered, it's just a 'bag of words' or a sort of histogram
- Rank by normalized scalar product between occurrence counts

Basically nearest-neighbor

$$sim(d_j, q) = \frac{\langle d_j, q \rangle}{\|d_j\| \, \|q\|} = \frac{\sum_{i=1}^{V} d_j(i)\, q(i)}{\sqrt{\sum_{i=1}^{V} d_j(i)^2} \sqrt{\sum_{i=1}^{V} q(i)^2}}$$

  - But this is sort of a slow approach.
  - But... we can train an SVM on this! See Caltech 101 dataset. They just used a linear SVM.
    - Make a histogram of visual words for a bunch of labeled objects → training data
    - SVM can then be trained to classify the object type
- Codewords are a common approach in CV today

8D-L1

- Introduction to Video Analysis
  - Image data here is now a function of (x,y) and t
  - 'activity recognition' goal is to recognize the activity of objects, so you first must find objects
- Background subtraction is used constantly here
  - Still hard, even with static camera
- Simple approach:
  - Estimate background at time t
  - Subtract estimated background from current input frame
  - threshold abs. difference to get foreground mask
- Frame differencing: $|I(x,y,t) - I(x,y,t-1)| > th$
  - Simple but not always useful. Large areas don't work well
- Mean filtering: Instead take mean of last n frames
  - Averaging isn't really 'right' here; it smears motion
- Now, realize that the background is basically constant most of the time, plus noise here and there.
  - So: Use the median. Longer n is fine here.
  - It has the advantage too that the background updates over time (not assumed constant)
  - Cons: Median needs much more memory (can't really do running median); need to set global threshold th
- Important paper here: 'Adaptive Background Mixture Models for Real-Time Tracking' (Stauer & Crimson)
  - They modeled background as a sum of Gaussians for cases (e.g. foliage, water) where backgrounds might vary but still be the background.

2015-11-11d

- Having a depth channel (e.g. Kinect) can really help with identifying background.
- But what if you have a static scene, but moving camera?
  - Think back to disparity & depth & parallax...
  - Just look at relative object motion. Larger motion = closer object
  - You end up with a volume of data and 'slices' in this along some axes give you sloping lines (e.g. XT slice)

Epipolar Plane Images, slide 23

8D-L2

- Activity Recognition
- The terms we'll use: (From Govindaraju & Bobick)
  - Event - single instant in time detection
  - Actions/Movements - atomic motion patterns
    - Gesture-like, clear-cut activity, nameable behavior (sit, wave arms)
  - Activity: series/composition of actions
- Basic approaches
  - Model-based action recognition
    - Human body tracking & pose estimation - relate to action descriptions
    - Major challenge: Getting good training data from contexts beyond testing

- Model-based activity recognition
  - Lower level detection of actions → activity, based on structural representation of activity
- Recently: activity as motion, space-time appearance patterns; typically we train a classifier on this, but no explicit body tracking
- What we're not covering:
  - Whatever, see slide 9
  - It's just the 3D extension of bag of visual words

slide 12
- Motion History Images (MHI)
  - Different function of temporal volume
  - Can be defined as recursive filter

  ] Can trivially turn to shorter time window lengths

- Motion Energy Image = thresholded MHI

- Old CV work that's still important:
  - Image Moments
    - Summarize a shape given image $I(x,y)$

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$

0th-order ($i=j=0$) is just area.

  - Central moments are translation-invariant:

$$\mu_{pq} = \sum_x \sum_y (x-\bar{x})^p (y-\bar{y})^q I(x,y)$$

$$\bar{x} = \frac{M_{10}}{M_{00}} \qquad \bar{y} = \frac{M_{01}}{M_{00}}$$

2011-11-11e

- But these moments aren't very robust to translation, rotation, scale
- Hu moments work around this
  - We pick 7, see slides 20 & 21
  - Translation, rotation, scale-invariant
  - This gives us $[h_1, h_2, h_3, h_4, h_5, h_6, h_7]$, a global space-time "shape" descriptor (if we do it for an MHI)
- How would you build a decent generative model of each class of action?
  - Maybe... Gaussian in Hu-moment feature space
  - Compare likelihoods: $p(\text{data} \mid \text{model of action } i)$
  - If have priors, use Bayes' rule:
    $$p(\text{model}_i \mid \text{data}) \propto p(\text{data} \mid \text{model}_i)\, p(\text{model}_i)$$
  - If not, use likelihood or even NN
- Some neat properties
  - It can proceed at any speed (look at min & max of training data)
  - Recursive formula → very fast
  - "Biological plausibility" (eyes & vision work similarly)
- See: KidsRoom (late '90s)
  - Millennium Dome

8D-L3

- Hidden Markov Models
- What can we do with time-series data?
  - Bird song spectrograms?
  - Stockmarket data?
  - Musical notation?
- How's this relate to vision?
  - Maybe we've a video and want to recognize gestures
- Markov Models
  - 1st order Markovian: Probability of moving to a given state depends only on current state.
  - States: $\{S_1, S_2, \ldots S_N\}$
  - Transition probabilities: $a_{ij} = P(q_{t+1} = S_i \mid q_t = S_j)$
  - Initial state distribution: $\pi_i = P[q_1 = S_i]$
  - With this you can ask things like...
    - How likely is the series $S_1, S_1, S_2, S_1, S_3, S_1$?
    - It's easily computed: $P(S_1)\,P(S_1|S_1)\,P(S_2|S_1)\,P(S_1|S_2)\,P(S_3|S_1)$
      - This tends to be a small number $P(S_1|S_3)$
      (N.B. Be careful when implementing. Use logarithms.)
- Hidden Markov Models
  - Maybe you can't observe the state, but can only observe evidence of the state
  - Emission probabilities: $b_j(k) = P(o_t = k \mid q_t = S_i)$
  - So we must ask, how likely is this series of evidence/observations?
    - But, this is a lot of things to add up... (slide 22)
    - All possible sequences matter.

2011-11-11f

- HMM specs: $\lambda = (A, B, \pi)$
  - N = # of states
  - $S = \{S_1, S_2, \dots S_N\}$ - Set of states
  - $Q = \{q_1, q_2, \dots q_T\}$ - Sequence of states
  - A = state transition probability matrix,
    $a_{ij} = P(q_{t+1} = j \mid q_t = i)$
  - B = observation probability distribution
    - Discrete: $b_j(k) = P(o_t = k \mid q_t = j), \ 1 \le k \le M$
    - Continuous: $b_j(k) = p(o_t = x \mid q_t = j)$
      (output is a distribution over some feature vector, e.g. in state 1 you have a measurement with one Gaussian, in state 2 a different Gaussian and so on)
  - π = the initial state distribution
    $\pi(j) = P(q_1 = j)$
- How this relates to vision:
  - Given sequence of observation, what <u>model</u> generated those?
  - the individual elements tell you something, but the time-series has sequentiality that matters

- 3 computational problems of HMMs.
  1. Evaluating $P(O \mid \lambda)$ - given model $\lambda = (A, B, \pi)$ what is the probability of occurrence of observation sequence $O = \{O_1, O_2, \dots O_T\}$?

→ Classification/recog. problem: With a trained model for each set of classes, which one probably generated what I saw?

2. Decoding: For $O = \{o_1, o_2, \dots o_T\}$ what optimal state sequence probably produced it?
   → Helps give meaning to states
3. Learning: Determine model λ, given training set of observations. (Find λ to maximize $P(O|\lambda)$)

- Back to problem 1: → depend only on state
  - Assume we know $Q = (q_1, \dots q_T)$, & independent observations
  - Then: $P(O|q,\lambda) = \prod_{t=1}^{T} P(o_t|q_t,\lambda) = b_{q_1}(o_1)\, b_{q_2}(o_2) \dots b_{q_T}^{(o_T)}$

    How likely is whole observation, given sequence q?

    How likely is observation o_t given state q_t?
  - That's trivial if we know the state sequence Q.
  - We know the probability of any given state sequence:

$$P(q|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} a_{q_2 q_3} \dots a_{q_{T-1} q_T}$$

    - π_{q_1}: Probability of starting in q_1
    - $a_{q_1 q_2}$: Probability of state $q_1 \to q_2$
    - and so on
  - Then...

$$P(O|\lambda) = \sum_{q} P(O|q,\lambda) P(q|\lambda)$$

    q: All state sequences.

    Yes, this is really, really inefficient.

    That's N^T states, each path 'costing' $T \to O(TN^T)$

2015-11-11 g

- Can we make that much more efficient? Yes!
- First: Define auxiliary forward variable α

$\alpha_t(i) = P(o_1, \dots o_t, q_t = i | \lambda)$

= probability of observing partial sequence $o_1, \dots o_T$ and state $q_t = i$ at time t

- This can be written recursively:
  - Forward recursive algorithm:
    - Init: $\alpha_1(i) = \pi_i b_i(o_1)$

↳ probability of observation o_1 at time 1, and state i

↳ Starting in i

↳ Seeing o_1 given you're in state i

- Each step: $\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(o_{t+1})$

↑ Transition (to j) probability

↳ prob. of being in state j, and having seen $o_1 \dots o_{t+1}$

Note that you could have reached j from any other state

just the $t+1$ output

- Conclude: $P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$ → Sum of observations and ending up in i, for all i
  - If you don't care about states, just observations
- This is $O(N^2 T)$ - much better.

- A backward recursive algorithm could compute likelihood of...
  - Being in i at time t
  - Observing remainder of observed sequence

] given HMM λ

(forward = observing sequence from start until t)

Where you see "sequence of discrete symbols", consider HMMs.

- If we know Hmm λ, either algorithm can estimate for us what state we're in at time t (giving a distribution)
  - With those distributions, one can find emission probs. $b_j(k)$ that maximize the sequence's probabilities.
    - And: One can find transmission probs. a_{ij} to maximize probabilities
  - New $a_{ij}(k)$ & $b_j(k)$ → New estimate of state distributions at all time.

The above is "expectation maximization"
(I don't quite get what this is for.

- HMMs = Generative probabilistic models of time series (w/ hidden state).
- Forward\backward algorithm for computing probabilities on hidden states
- See 'conditional random fields' for something working well for time-series - but much more complex.

- HMMs are often applied for gesture recognition.
  (There's a whole conference on Face & Gesture Recognition)
- Gestures had new-found life for human-robot interaction (esp. for noisy environments where speech is too hard)
- And: Hand gestures w/ data glove
  - Many slides on this...

2015-11-11h

- Hmm pros:
  - Does feature selection on spatial & temporal models
  - Recognition is fast, training's not
- Cons
  - Not good for on-the-fly labeling (e.g. segmenting input streams)
  - "Works well when problem is easy" - when regularities are very clear - and less easy/clear outside.

2015-11-21

9A-L1

- Color spaces
  - Dr. Bobick does this more from a psycho-physics perspective - that color is perceptual
- <u>Cones</u> in your eyes see high-res & color.
  - We call them red, green, blue, but that's not exactly accurate (these are ~ 64% 'red', 32% 'blue', 2% 'blue')
  - See normal perceptual chart, slide 3
  - Hardly any blue cones are in the <u>middle</u> of retina

2015-11-22

- Slide 8 has some 'negative red' thing I don't quite get
- Why red & green together produce yellow is psychological. The spectrums don't 'average' them.

- It relates to cones, not physics. Red cones & green cones both respond, and we can't actually tell this apart from yellow light.
- We deal much better with luminance than color differences.
- CIE RGB, CIE xy chromaticity diagram
- CIE XYZ color space
  - Linear transform of CIE RGB
  - Y = perceived luminance, X/Z = perceived color
  - $x = \frac{X}{X+Y+Z}$, $y = \frac{Y}{X+Y+Z}$ (XYZ → xy chromaticity)
- CIE L*a*b* (not Lab)
  L = luminance, a/b produce different colors (hues?)
  (a,b) is 'purer' as you move toward outside
- Dr. Bobick prefers the cone, not cylinder, representation of HSL. It narrows to a point at the top (highest lightness) and bottom (lowest lightness), showing that the darker or lighter a color is, the less room there is to talk about its color (white & black have no hue).
- What we know best is RGB color space
- Color gamut: a subset of possible colors
  - See: ProPhoto RGB, Adobe RGB, sRGB (most monitors)
    - ProPhoto & Adobe are much larger.
    - You must sometimes figure out what to do with colors that are outside of the gamut you are mapping into

2015-11-21 (b)

- When we're dealing with pixels in most of our images, each pixel is a <u>vector</u> of (r,g,b). We can plot these in (r,g,b) space rather than (x,y) if we want, image space → color space
  - Note on slide 30 that the color plot has some clearly segmented bits of color
- It helps to separate intensity & color
  e.g. $Y = 0.299R + 0.587G + 0.114B$
  RGB combines intensity <u>and</u> color in all 3 values.
  YUV separates this out somewhat.
  $U \approx 0.492(B-Y)$, $V \approx 0.877(R-Y)$
  Assume $R, G, B, Y \in [0,1]$
  (For better algebraic version see slides 36 & 37)
- (Y,U,V) plot, or (Y,U,V) projected to (U,V), shows color clustering much more clearly
- Why YUV?
  - Easier pixel clustering (minor reason though)
  - <u>Chroma subsampling</u> for efficient encoding!
    - Use more bits for Y, less for U & V because we are more sensitive to Y (luminance)
    - e.g. YUV 422 - twice as much for Y as for U or V
- Look at a low-light image/video once after it's compressed, and look at blue channel. It will have almost all the noise because very few bits are used here.

9A-L2

- Segmentation
  - Segmenting into regions of interest (and what this means depends on usage)
  - Berkeley segmentation database has a bunch of human-drawn segmentations of photos
- One kind: figure-ground segmentation
- Or: "superpixels" which fragment an image up, such that each fragment sits within the same segment (which greatly simplifies later segmentation)

slide 5

- We often need to do <u>clustering</u> to perform segmentation.
- Good clusters minimize SSDs between all points & their respective sensors. Now, how do we choose cluster centers? Without just already knowing the clusters...
- So... <u>K-means clustering</u>, an iterative algorithm.
  1. Randomly initialize cluster centers $c_1, \ldots c_K$ (You might also do it smarter than just 'random')
  2. Find points in each cluster:
     - For each point p, find closest c_i; put p into cluster i
  3. Given points in each cluster, solve for c_i:
     - Set c_i to be <u>mean</u> of points in cluster i.
  4. If any c_i changed, repeat step 2.

Why mean? Mean can be shown to minimize the SSD.

- Good example, slides 15-19
- If we're doing this for segmentation, the <u>feature space</u> we work in matters (e.g. for B&W image - pixel intensity?)

2015-11-21c

- One problem with k-means is that you must know the value of k (how many clusters to find).
- It can be seen as a k-level quantization of the feature space
- Color-based clustering can be useful too
- Or: Cluster based on intensity and position.

slide 25

(For (x, y, I), that's a 3D feature space.
Or: (x, y, r, g, b), a 5D space.)

- K-Means pros: Simple, guaranteed to converge to local minima
  Cons: Very memory-intensive (must hold all points)
  Must pick k
  Sensitive to initialization
  Sensitive to outliers
  Biased toward "spherical" clusters, even if it means breaking up 'closer' clusters and joining distinct ones.

9A-L3

- Mean shift segmentation
  - This looks for 'modes' or local maxima of density.
- Basic idea:
  - Start with your distribution of points in (feature) space
  - Have some defined region of interest (Gaussian-weighted is good - you want falloff at the edges)
  - Find region's center-of-mass. Move the region's center to it. Do this until it doesn't move!

↳ region for which all trajectories lead to the same mode

- Cluster: all data points in the attraction basin of a mode.

- To use mean-shift for segmentation:
  - Find features (colors, gradients, texture, etc.)
  - Init windows at individual feature points (pixels?)
  - Perform mean-shift at each window (pixel?) until convergence
  - Merge windows (pixels?) that end up near same 'peak'/mode
- See slides 13, 14, 15 - it works well, but depends on feature space. It may however give results that are more like 'superpixels' to reduce the units we must deal with to something less granular than pixels.
- Mean-shift pros:
  - Automatically finds basins of attraction
  - One parameter choice: window size
  - Assumes no shape on clusters
  - Very generic
  - Finds multiple modes
- Cons:
  - Must choose window size and this can be tricky
  - Scales badly with dimensionality of feature space.

- Color, brightness, position - usually insufficient alone to distinguish regions. We use texture extensively too.
- But how do we bring texture into our feature space?

slide 19

  - One way: A bank of filters that cover (e.g.) different scales, angles, and the filter response of each is a dimension.

2015-11-21d

- Find "textons" by clustering vectors of filter bank outputs
  - Describe texture in window based on "texton histogram" (Reminds me of SIFT a little?)
  - That is, the window is a little larger - and a texture is composed of many different textons in some distribution.
- The combination of this with other segmentation can be very powerful

9A-L4

- Segmentation by graph partitioning
  - We can view images as <u>graphs</u>. The pixels are connected to their neighbors.
  - There is a <u>node</u> for every pixel, and a link for every <u>pair</u> of pixels $\langle p, q \rangle$.
  - Each linkledge has affinity weight w_{pq} which measures <u>similarity</u>.
- One way (a simple one):

$$aff(x_i, x_j) = \exp\left(\frac{-1}{2\sigma^2} dist(x_i, x_j)\right)^2$$

- Then segmentation is just breaking graphs into segments
  - Easiest links to break: lowest affinity
- <u>Graph cut</u> is what's needed, it's a set of edges whose removal makes a graph disconnected.
  Cost of cut: sum of weights of cut edges

$$cut(A,B) = \sum_{p \in A,\ q \in B} w_{pq}$$

- So, what's a 'good' graph cut & how do we find one?
  - See min-cut algorithm (or max-flow - they're the same)
- Problem: min-cut looks for smaller cuts (in terms of number of edges) which makes small, isolated graphs
- Fix bias w/ normalization for size of segments:

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$

$assoc(A,V)$ = sum of weights of all edges touching A
- this is sort of how "well-connected" components A & B are. It penalizes smaller ones.
- Approx. solution (exact is hard): Generalized eigenvalue problem
  - Let W = adjacency matrix of graph
    D = diagonal matrix with diagonal entries
    $D(i,i) = \sum_j W(i,j)$ (the 'degree' of the node, how many nodes are connected to it?)
  - Normalized cut cost: $\frac{y^T (D-W) y}{y^T D y}$

y = indicator vector, has 1 in the i^{th} position if i^{th} feature point belongs to A, otherwise, -C (or -1)
- That's very hard to solve for y.
- But... Solve $(D-W)y = \lambda Dy$ for eigenvector with 2nd smallest eigenvalue. Use eigenvector entries to bipartition graph.
  This is an approximate solution.

2015-11-24

- See: Berkeley segmentation engine.
- To get more segments (bipartition gives you 2): just apply recursively.
- Normalized cuts, pros:
  - Very generic. Flexible to choice of function that computes affinities between nodes.
  - Doesn't need data distribution model
  - Graphs are well-understood, well-studied!

separates affinity calculation & segmentation

- Cons:
  - High time complexity. Matrices can be huge, NxN for N elements in feature space.
    - Dense, highly-connected graphs — many affinity computations. It is worse with larger regions.
    - Must solve eigenvalue problem.
  - Prefers balanced partitions. Not always appropriate.
- Overall, though, works very well.

2015-11-29

9B-L1

- Binary morphology
  - these aren't 'current' or 'state of the art' but are very important
  - It comes up anyplace images do that are only 0s and 1s
  - The morphological stuff we did is this

- Typ. 0 = background, 1 = foreground (not-stuff vs. stuff)
- Operations:
  - Separate objects from background & each other
  - Aggregate pixels, compute features, count objects
- Example: Counting red blood cells
- From a greyscale image:
  - Thresholding, finding good threshold
  - Connected components analysis
  - Binary mathematical morphology
  - Feature extraction, statistics
- Finding threshold is non-trivial but one method is to use histogram - see Otsu's method
  - That assumes bimodal histogram
  - It finds threshold t minimizing the weighted sum of within-group variances for two groups that result from separating grey tones at t.
- While it assumes bimodal it may be done recursively to detect further modes. If variance barely changes then it may signify no separate mode there.
- Connected components methods - slide 13
- N.B. Difference with 4-way connected & 8-way connected:

0	1	0
1	1	1
0	1	0

4-way

1	1	1
1	1	1
1	1	1

8-way

2015-11-29 (b)

- Morphology operations: Dilation & Erosion
  - Composite operations based on these:
    - Closing & opening
    - Thinning, thickening

$A \oplus B$

- Dilation: Expands connected sets of 1s
  - Grows features, fills holes & gaps.

$A \ominus B$

- Erosion: Shrinks connected sets of 1s.
  - Shrinks features, removes bridges, branches, protrusions
- Structuring element: Shape mask used in basic morph. ops. It has a defined origin (typ. center).
  - It has 0s, 1s, and sometimes X (don't care) but we don't deal with that here.

Slide 25

- Dilation, with binary image B & element S:
  - Put origin of S at each pixel of B, and at every 1 location of S, compute OR w/ corresponding B element
- Erosion: Same thing but AND, not OR
  - This is easy to think about as you are merely looking for the same pattern of 1s as in S.
- Most useful morphological ops: Opening & Closing
  - Opening: Erosion, then dilation (w/ same structuring element)
    - It can be shown that opening of A by B = union of all translations of B that entirely fit within A. Intuitively: If we have a B-shaped brush, it is what we can paint if we can't go outside A.

- Math here is rather abstract and relates to set theory
  - It's not really a CV sub-field anymore
- It is idempotent - repeating it has no further effect
- Closing: Dilation, then erosion.
  - Complement of union of all translations of B that don't overlap A.
  - Intuitively: If we've a B-shaped brush and cannot paint inside A - it's what we ~~can't~~ can paint
- Also idempotent!
- Boundary of A: $A - (A \ominus B)$
- Thinning: $A \otimes B = A - (A \circledast B) = A \cap (A \circledast B)^c$ → 'Hit or miss' operator?
- Thickening: $A \odot B = A \cup (A \circledast B)$
- Morphology is powerful with clean & almost-clean binary images (which are easily cleaned)

9C-L1

- 3D Perception
  - Monocular vision is very challenging to sense 3D with
- Passive 3D sensing: Adds no illumination. Our stereo matching was this.
  - Shape from focus is another
  - Exploit known geometry or properties
- Active 3D sensing: Project light/sound into environment, see how it reacts; maybe encode some pattern which sensors can pick up
  - Photometric stereo, TOF (ultrasound, LIDAR)
  - Structured light (basically stereo in another form)
- Lots of stuff here is what I had already read of 2 years ago on PrimeSense...

2015-11-29c

- Slides 33-35 - basic PrimeSense algorithm
- Pros of IR:
  - Works in low-light!
  - Doesn't rely on textured objects
  - Repeated scene textures don't cause issues
- Tailor algorithm to produced pattern
- Pros of natural light:
  - Works outside!
  - Resolution not limited to projector
- Cons of both:
  - Dark surfaces can cause problems
  - Specular reflection does too
- N.B. Depth images don't just tell you surfaces, they tell you about <u>lack of occlusion</u> between camera & surface!
- But they are viewpoint-dependent.
- So: Point clouds. (They lose occlusion info but are viewpoint-independent)
  - Disadvantages (also): Sparse; variable density (based on distance from sensor)
  - Biggest advantage: PCL, Point Cloud Library
  - Point clouds are <u>sampled</u> from object surface. Volume & connectivity must be derived!
- Point clouds (so he claims) <u>have</u> surface normals built in (see slide 50)
- Similar technique to SIFT can be used for point clouds (if it can be made translation <u>and</u> orientation invariant) to match details.

e.g. 5x5x5 histogram

"Fast Point Feature Histogram"

~30% of cortex is related to vision

2015-11-30

10A-L1

- The Retina
  - Monocular vision: 160° (horizontal), each eye
  - Binocular vision: 120° (horizontal)
  - Total visual field: 200° (horizontal) x 135° (vertical)
- Right visual field of both eyes goes to left half of brain. Likewise, left visual field to right half of brain.
- Hemifield neglect: An injury to left side of brain may lose right half of vision - but you don't just see a hole, your brain just sort of ignores it and fills in.
  - Your own blind spot is much the same
- Humans have 'inverted retina' - light passes through other cells & blood vessels, then rods & cones. Yes, these sometimes produce shadows - but the brain removes things that are stationary on the eye.
- When we speak of one's 'eyes adjusting', this isn't the pupil opening - that's near instant. It is your photoreceptors changing sensitivity.
- A rod can detect even a single photon.
- In fovea (middle 1-2° of eye) is a tight hexagonal grid of cones, at periphery it has larger photoreceptors (cones) in layout.

  Your peripheral vision is very low-resolution, but senses motion well.
- Your fovea cannot see low-light, but periphery can!

  You can notice with stars at night sometimes

2015-11-30 (b)

- Eye handles 10^{-6} to 10^{+8} cd/m^2 (which is huge)
- HDR tries to get this same range
- It's not the pupil, but the retina's ganglion cells that make this work. Pupil gets only 10-100 : 1.
- They can actually do this independently (see center-surround receptive fields)

10B-L1

- Human Vision System: Vision in the Brain
- He recommends a course in Sensation Perception and Neuroanatomy if this topic interests you
- This reminds me a bit of what Dr. Kabrisky talked about at Honors Seminars maybe 10 years ago (11? 12?)
  - Slide 7 is what looks familiar

- Instructor cautions us: This felt like a comprehensive course but it's very 'careful' with what images it uses. 'Real-world' stuff has much more to deal with.

2015-10-08

"The Laplacian Pyramid as a Compact Image Code"
(Burt & Adelson, 1983)

Reduce: $g_\ell(i,j) = \sum_{m=-2}^{2}\sum_{n=-2}^{2} w(m,n)\, g_{\ell-1}(2i+m, 2j+n)$

Expand: $g_{\ell,n}(i,j) = 4\sum_{m=-2}^{2}\sum_{n=-2}^{2} w(m,n)\cdot g_{\ell,n-1}\left(\frac{i-m}{2}, \frac{j-n}{2}\right)$ ← only for integral indices

$g_{\ell,n}$ = result of expanding g_ℓ n times.
So $g_{\ell,0} = g_\ell$; $g_{\ell,n} = \text{EXPAND}(g_{\ell,n-1})$

- Weighting kernel needs to be: separable, normalized (in one dimension), symmetric
  - They use: $w(m,n) = \hat{w}(m)\,\hat{w}(n)$
    $\hat{w}(0) = a$, $\hat{w}(-1) = \hat{w}(1) = 1/4$, $\hat{w}(-2) = \hat{w}(2) = 1/4 - a/2$

(Why?)
(I guess it's the only solution to all constraints...)

So for expand function...

If i,j even: $\frac{i-m}{2} = \frac{i}{2} - \frac{m}{2}$, even when m even
$\frac{j-n}{2} = \frac{j}{2} - \frac{n}{2}$, even when n even

i odd: $\frac{i-m}{2}$ is integer when m odd

j odd: $\frac{j-n}{2}$ integral when n odd

i,j

even, even

$$w(-2,-2)\,g\left(\tfrac{i+2}{2}, \tfrac{j+2}{2}\right) + w(-2,0)\,g\left(\tfrac{i+2}{2}, \tfrac{j}{2}\right) + w(-2,2)\,g\left(\tfrac{i+2}{2}, \tfrac{j-2}{2}\right) +$$
$$w(0,-2)\,g\left(\tfrac{i}{2}, \tfrac{j+2}{2}\right) + w(0,0)\,g\left(\tfrac{i}{2}, \tfrac{j}{2}\right) + w(0,2)\,g\left(\tfrac{i}{2}, \tfrac{j-2}{2}\right) +$$
$$w(2,-2)\,g\left(\tfrac{i-2}{2}, \tfrac{j+2}{2}\right) + w(2,0)\,g\left(\tfrac{i-2}{2}, \tfrac{j}{2}\right) + w(2,2)\,g\left(\tfrac{i-2}{2}, \tfrac{j-2}{2}\right)$$

odd, even

odd, odd

$$= w(-2,2)\,g(i'+1, j'+1) + w(-2,0)\,g(i'+1, j') + w(-2,2)\,g(i'+1, j'-1) + w(0,-2)\,g(i', j'+1) + w(0,0)\,g(i', j') +$$
$$w(0,2)\,g(i', j'-1) + w(2,-2)\,g(i'-1, j'+1) + w(2,0)\,g(i'-1, j') + w(2,2)\,g(i'-1, j'-1)$$

$$g_{\ell,i}(i,j) = 4\sum_{m=-2}^{2}\sum_{n=-2}^{2} w(m,n)\, g_{\ell,n-1}\left(\frac{i-m}{2}, \frac{j-n}{2}\right)$$

where $g_{\ell,n-1}(x,y) = 0$ for non-integral x or non-integral y.

Or: $g'_{\ell,n-1}(x,y) = 0$ for odd x or odd y,

and $g_{\ell,n-1}\left(\frac{i-m}{2}, \frac{j-n}{2}\right) = g'_{\ell,n-1}(i-m, j-n)$.

Then: $$g_{\ell,n}(i,j) = 4\sum_{m=-2}^{2}\sum_{n=-2}^{2} w(m,n)\, g'_{\ell,n-1}(i-m, j-n)$$

This is simply convolution, and $g'_{\ell,n-1}(x,y) = g_{\ell,n-1}\left(\frac{x}{2}, \frac{y}{2}\right)$ which is easily achieved simply by doubling $g_{\ell,n-1}$ in size - but leaving its values only at even indices, and assuming that all others are zero.

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} a & -b & c \\ b & a & d \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$$u' = au - bv + c$$
$$v' = bu + av + d$$

$\rightarrow$ $Ax = B$,

where $A = \begin{bmatrix} u_1 & -v_1 & 1 & 0 \\ v_1 & u_1 & 0 & 1 \\ \vdots & & & \end{bmatrix}$

$x = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$

$B = \begin{bmatrix} u' \\ v' \\ \vdots \\ \end{bmatrix}$

Problem 3b (PS5) gives similarity transform:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} a & -b & c \\ b & a & d \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Multiplying out: $u' = au - bv + c = au + (-b)v + c + 0$

$v' = bu + av + d = av + bu + 0 + d$

This can be written as $Ax = B$ where:

$x = [a \;\; b \;\; c \;\; d]^T$ (4×1 matrix)

$B = [u_1' \;\; v_1' \;\; u_2' \;\; v_2' \;\; \cdots \;\; u_n' \;\; v_n']^T$ ($2N \times 1$ matrix)

$$A = \begin{bmatrix} u_1 & -v_1 & 1 & 0 \\ v_1 & u_1 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ u_n & -v_n & 1 & 0 \\ v_n & u_n & 0 & 1 \end{bmatrix}$$

($2N \times 4$ matrix)

Affine (from 3c): $\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$

$u' = au + bv + c$

$v' = du + ev + f$

$Ax = B, \quad x = [a, b, c, d, e, f]^T \quad (6 \times 1)$

$B = [u'_1, v'_1, u'_2, v'_2, \ldots, u'_n, v'_n]^T \quad (2N \times 1)$

$$A = \begin{bmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_1 & v_1 & 1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 \\ \vdots & & & & & \\ u_n & v_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_n & v_n & 1 \end{bmatrix} \quad (2N \times 6)$$

---

translation: $s = 1$

similarity: $s = 2$

affine: $s = 3$

Arbitrarily say that $p = 0.99$, $\epsilon = 0.10$ (99% chance of success, 10% outliers)

Then for translation, $N > \frac{\log(0.01)}{\log(1 - 0.90)} \Rightarrow N \geq 2$

For similarity, $N > \log(0.01) / \log(1 - 0.90^2) \Rightarrow N \geq 3$

For affine, $N > \log(0.01) / \log(1 - 0.90^3) \Rightarrow N \geq 4$

If $p = 0.999$, $\epsilon = 0.20$, $N \geq 5$, $N \geq 7$, $N \geq 10$, respectively.

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-x^2/2\sigma^2}$$

$$G(x,y) = G(x)G(y) = \frac{1}{\sqrt{2\pi\sigma^2}}\left(e^{-x^2/2\sigma^2} e^{-y^2/2\sigma^2}\right) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x^2+y^2)/2\sigma^2}$$

Let $G'(x) = \frac{\partial}{\partial x} G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \frac{\partial}{\partial x}\left(e^{-x^2/2\sigma^2}\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \frac{\partial}{\partial x}\left(\frac{-x^2}{2\sigma^2}\right) e^{-x^2/2\sigma^2}$

$= \partial/\partial x\left(\frac{-x^2}{2\sigma^2}\right) G(x) = \frac{-2x}{2\sigma^2} G(x) = \frac{-x}{\sigma^2} G(x)$

Let $G_x(x,y) = \partial/\partial x\, G(x,y) = \partial/\partial x\, G(x)G(y) = G(y)\,\partial/\partial x G(x)$

$= G(y)\,G'(x) = \frac{-x}{\sigma^2} G(x)G(y)$

Likewise $G_y(x,y) = \partial/\partial y\, G(x,y) = G(x)\,\partial/\partial y\, G(y) = \frac{-y}{\sigma^2} G(x)G(y)$

Or: $G_x(x,y) = G'(x)G(y)$

$G_y(x,y) = G(x)G'(y)$

] Thus, both are Separable.

- OpenCV: See sepFilter2D\filter2D